FORCE on Nextflow: Scalable Analysis of Earth Observation Data on Commodity Clusters

Fabian Lehmann[1], David Frantz[1], Sören Becker[2], Ulf Leser[1] and Patrick Hostert[1]

[1]Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany
[2]Technische Universität Berlin, Straße des 17. Juni 135, 10623 Berlin, Germany

Abstract

Modern Earth Observation (EO) often analyses hundreds of gigabytes of data from thousands of satellite images. This data usually is processed with hand-made scripts combining several tools implementing the various steps within such an analysis. A fair amount of geographers' work goes into optimization, tuning, and parallelization in such a setting. Development becomes even more complicated when compute clusters become necessary, introducing issues like scheduling, remote data access, and generally a greatly increased infrastructure complexity. Furthermore, tailor-made systems are often optimized to one specific system and cannot easily be adapted to other infrastructures. Data Analysis Workflow engines promise to relieve the workflow developer from finding custom solutions to these issues and thereby improve scalability, reproducibility, and reusability of workflows while reducing development cost at the infrastructure side. On the other hand, they require the workflow to be programmed in a particular language, to obey certain principles of distributed processing, and to properly configure and tune the execution stack, which puts additional burden to data scientists.

Here, we study this trade-off using a concrete EO workflow for long-term vegetation dynamics in the Mediterranean. The original workflow was programmed with FORCE, a custom-made framework for assembling and executing EO workflows on stand-alone servers. We ported it to the scientific workflow system Nextflow, which is capable of seamlessly orchestrating workflows over a large variety of infrastructures. We discuss the pitfalls we faced while porting the workflow, advantages and disadvantages of such an approach, and compare in detail the efficiency of both implementations on various infrastructures. We quantify the overhead in execution time incurred by the workflow engine and give hints on how to deal with heterogeneous tasks. Overall, our Nextflow implementation shows promising behavior in terms of reusability and scalability, though this does not apply to all workflow stages.

Keywords

FORCE, Nextflow, Workflow porting, Scaleability, Reproducibility, Earth observation, Landsat

1. Introduction

Developing, modifying, and executing workflows is the daily business of many EO scientists. Those workflows typically deal with large amounts of input data passed through a sequence of different tools transforming and extracting valuable insights from the data. Due to the sheer amount of data and the complexity of some of the processing steps, EO workflows are rather resource hungry; at the same time, the tools involved exhibit very heterogeneous requirements in key factors such as memory, I/O performance, or compute power.

Building infrastructures for designing and executing such workflows is demanding. It requires a deep understanding of parallelization strategies, tool synchronization, scheduling, and data transport mechanisms. Things get even more complex when a shared-nothing distributed system should be used. Although custom-made systems are still very popular, they face the additional challenge that they are often optimized for a specific environment and accordingly not or only with large effort portable to other systems. This impedes reusability and reproducibility [1, 2].

Data Analysis Workflow (DAW) engines like Nextflow [3], Airflow [4], or Pegasus [5] promise to reduce the complexity by providing automatic parallelization, distribution, and scalability over large clusters, improved reproducibility and reusability, and generally reduced development cost. However, they require a steep learning curve, which might look rather unattractive for a domain scientist interested in analyzing only a concrete data set. In this paper, we study the trade-off between the effort necessary to port an EO workflow from a custom solution to a DAW engine and the benefits one can harvest once the port is done. Thereby, we do not change the software used. We report typical problems one faces in such a transformation and analyze the behavior of the different approaches on different infrastructures in detail.

Although the choice of infrastructure for running DAWs over large scientific data sets is a decision with long-lasting impact, there are comparably few works that

CDCEO 2021: 1st Workshop on Complex Data Challenges in Earth Observation, November 1, 2021, Virtual Event, QLD, Australia.
fabian.lehmann@informatik.hu-berlin.de (F. Lehmann); david.frantz@uni-trier.de (D. Frantz); soeren.becker@tu-berlin.de (S. Becker); leser@informatik.hu-berlin.de (U. Leser); patrick.hostert@geo.hu-berlin.de (P. Hostert)
0000-0003-0520-0792 (F. Lehmann); 0000-0002-9292-3931 (D. Frantz); 0000-0001-6487-1268 (S. Becker); 0000-0003-2166-9582 (U. Leser); 0000-0002-5730-5484 (P. Hostert)

provide guidance and benchmarks [2, 6, 7, 8, 9]. Such comparisons also quickly become outdated as infrastructures and systems evolve. Here, we want to contribute to closing this gap for the particular case of EO workflows.

Our paper is structured as follows: Section 2 presents FORCE and the original workflow. Section 3 describes Nextflow, and Section 4 discusses important design decisions that had to be taken for porting the workflow. In Section 5, we outline our experiments and Section 6 presents a quantitative evaluation of both approaches. Our findings are discussed in Section 7. The original and the ported workflow can be found at GitHub[1].

2. FORCE and the EO workflow

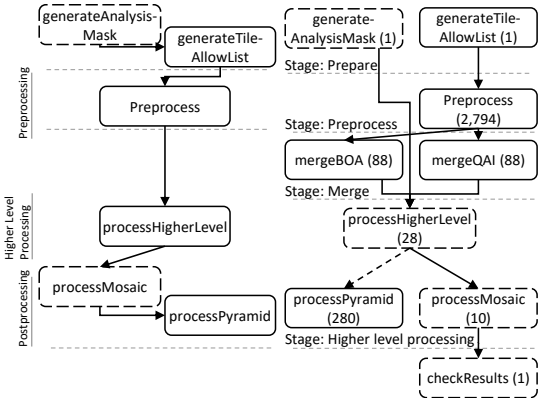
FORCE[2] (Framework for Operational Radiometric Correction for Environmental monitoring [10]) is an all-in-one solution for the analysis of large volumes of medium-resolution EO data. FORCE was initially developed as a means to reduce the entry barrier of large-scale EO analyses by providing a rich and integrated toolset that may be linked with operational monitoring systems [11]. FORCE consists of various tools that can be assembled into complex EO workflows, including methods for data download, image preprocessing, data aggregation/reduction, and analyses techniques based on machine learning or time series analysis. FORCE workflows are configured via parameter files to define processing options, input/output locations, parallelization parameters, etc.

Long-term vegetation dynamics on Crete

In this paper, we focus on a specific workflow to re-assess the widespread rangeland degradation in the Mediterranean as reported 20 years ago with limited input data [12, 13]. With the unlimited data access of today, our analysis shows that total vegetation on the island of Crete, Greece, did rather increase. Yet, we still cannot dispel that vegetation degradation occurred as most increase in vegetation cover was found in the woody vegetation, which potentially represents a degradation process related to the increase of impalatable species.

Data: For the analysis in this paper, we used 304GB of input data covering the island of Crete. We leveraged multispectral data from the Landsat mission, which observes the land surface at 30m spatial resolution each 8–16 days. In total, 2,794 images were retrieved[3] (300GB). We downloaded all available L1TP/T1 data from Landsat 4, 5, and 7 for the years 1984 – 2006 with a cloud coverage of less than 70%.

[1] https://github.com/CRC-FONDA/FORCE2NXF-Rangeland
[2] https://github.com/davidfrantz/force
[3] https://console.cloud.google.com/storage/browser/gcp-public-data-landsat



(a) FORCE workflow DAG (b) NextFlow workflow DAG

Figure 1: DAGs: Boxes represent processes and arrows their execution order (a) or mutual dependencies (b). Solid arrows mean that all parent tasks/the parent process must finish completely before dependent task can start, whereas dashed arrows indicate that a dependent task can start as soon as its parent task has been processed. Solid boxed mark CPU-, dashed boxes IO-bound tasks. Numbers in brackets represent the number of executions.

The 1-arcsecond SRTM Digital Elevation Model (DEM: 1GB) was used for preprocessing. Gaps were filled using the ASTER DEM. A pre-compiled water vapor database [14, 15] was used for correcting gaseous absorption during atmospheric correction (3GB).

Original workflow

The original workflow consists of multiple sequential processing stages (Figure 1a). Processing within stages is parallelized. All tools involved are executed in Docker containers orchestrated through a Bash script.

1) **Preprocessing**. All input images are converted to Level 2 Analysis Ready Data (ARD [10]) with corrections for atmospheric, topographic, adjacency, and BRDF effects, as well as cloud/shadow detection [16, 17, 18, 19, 20, 21]. Corrected images are reprojected to a shared projection (EPSG:3035) and split into image chips according to a regular 30km x 30km reference grid, thus forming a tiled data cube structure. FORCE processes multiple input images in parallel.

2) **Higher Level Processing**. For each ARD image, fractional vegetation cover is derived [22, 23], followed by noise-based [24] outlier detection and inlier restoration, as well as temporal interpolation [25]. Phenological metrics are acquired [26] to decompose the time series into woody and herbaceous vegetation cover, followed by a change and trend analysis [27]. As higher level processing is often I/O-bound and memory limitations might occur, FORCE sequentially processes the tiles with a nested multithreading parallelization strategy (OpenMP), wherein image blocks are processed in

sequential order with three threading pools taking care of input, compute, and output, respectively.

3) **Postprocessing**. To facilitate and accelerate visual interaction, the last stage of the workflow generates image pyramids and virtual mosaics for each output image.

The workflow produces in total 29GB of output, which includes images of interpolated time series, polar-transformed time series, time series of phenological metrics, change and trend parameter images - along with corresponding image pyramids and virtual mosaics.

After running the workflow, we performed a classification of change and trend parameters into broad land change categories. Area statistics of this map were used for testing reproducibility during our porting.

3. Nextflow workflow engine

When executing a workflow, FORCE directly manages task parallelism and scheduling. Execution is expected to run on a single server, meaning that tasks may share memory and write to the same disk. In a workflow system like Nextflow, the situation is rather different. Resources on remote machines, as well as remote data access, must be carefully managed, and a scheduler must decide which tasks to put on which compute node. In our installation of Nextflow, these duties are shared among three different independent software systems: we use Kubernetes for distributed resource management, Nextflow for orchestrating and scheduling task executions, and Ceph as a distributed file system. We briefly describe Nextflow in the next paragraphs and refer the reader to the Kubernetes documentation[4] and [28] for details on Ceph.

Nextflow[5] is a rather recent, domain-agnostic DAW engine consisting of a workflow language, the workflow engine, and a set of connectors to run workflows on different infrastructures, including Kubernetes [3]. It emerged from Bioinformatics applications but also finds increasing uptake in other scientific domains and in industry.

Nextflow workflows consist of channels (for data exchange) and processes (for task execution). Processes may wrap any command-line tool, offering the option to include tasks implemented in different languages into the same workflow. Channels define dependencies between processes by means of provider/consumer relationships at the file level. Files are passed into the according output channel if a process' output matches a specified pattern. Once all output channels that serve as input channels for a subsequent process are filled, the respective task is ready for execution and gets scheduled by the workflow engine. As a result, Nextflow automatically executes tasks in parallel and in a distributed manner.

[4] https://kubernetes.io/docs/home/

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

In Kubernetes, Nextflow first starts a driver pod which spawns pods for each workflow task. All pods mount the same, POSIX compatible, shared file system (Ceph) for storing and exchanging data. Each pod executes its task in a local, temporary directory and copies the output to the shared file system only upon completion.

4. Bringing FORCE into Nextflow

Porting the FORCE EO workflow (Section 2) to our Nextflow installation was mostly straight-forward. Tools from FORCE are wrapped as processes in Nextflow processing only one task at a time, while the dependency structure of the workflow is modeled with channels between tasks. The resulting Nextflow workflow comprises nine tasks (Figure 1b). However, the distributed execution environment brings a number of (sometimes subtle) incompatibilities that required adaptation of the FORCE logic.

Synchronized file access: To reduce data redundancy, FORCE, during preprocessing, merges files representing parts of the same observation. Specifically, FORCE replaces two images by their average in case of reflectances or by the latest for quality images. Since images are compressed, they first must be read and then written. To avoid data loss or corruption, the access is coordinated through lock files. This, however, is impossible in a shared-nothing distributed setting as access may happen on different machines.

In the Nextflow workflow, we changed this pattern to make tasks independent - and thus parallelizable - from each other (see tasks in the same horizontal level in Figure 1b). To retain the original functionality, we subsequently group all images by their tile, date, and satellite, and then merge every group in a separate, newly introduced process (see merge step in Figure 1b). This procedure improves scalability and flexibility regarding the execution environment but also generates more I/O since files potentially have to be moved to different nodes.

This change, however, created a new issue. When implementing each merge step as an individual task, we observed very inefficient workflow execution. The reason is that each of these tasks requires only a few seconds but must be started and configured anew by Nextflow for every execution, leading to start-up times being higher than execution times. As this task is executed thousands of times, performance degraded considerably. As a workaround, we decided to execute batches of size 100 as a single process in Nextflow, reducing the cumulative start-up times accordingly. This batch size is another trade-off in the new system: larger batches reduce parallelism but also wastage through process start-ups. We grouped these tasks by tile since all images of the same tile are processed in the same succeeding task.

Working around relative file structures: A signifi-

cant difference between FORCE and Nextflow is the way they deal with directory hierarchies. In FORCE, the data are structured in directories such that observation time and the observing satellite are encoded in the filename, whereas location is encoded in the directory name (as tile ID). In contrast, Nextflow only works on the file level and cannot easily cope with directories. As a remedy, we rename the output images of the preprocessing task by prefixing them with the tile ID. We clip this prefix for the higher level process in a custom wrapper and create a directory as expected by the task. Clearly, such tricks do not make a workflow easier to understand.

5. Evaluation setup

We performed two types of experiments to investigate whether the ported workflow scales as expected and to detect potential bottlenecks. We first ran the original workflow in its original environment to obtain confirmed results and ensured that all other configurations produce the same results. We subtracted the runtime of the check-result task in the Nextflow workflow from the overall execution time, to achieve comparable results.

Experiments were repeated three times; we report the median of the measured runtimes. We measure wall-clock execution times. For the distributed setting, we also report on efficiency of task executions, defined as the theoretical time obtained by dividing single node execution time through the number of nodes, divided by the observed runtime. Thus, an efficiency of 1 means perfect scaling, while 0.5 means that the distributed runtime is only half as good as theoretically possible. We utilized Nextflow version 21.04.0-edge with bugfixes[6][7] in the cluster with Kubernetes version 1.19.3 and Ceph version 15.2.8, and Nextflow version 20.10.0 locally.

Experiment 1 (Single server performance): As the original workflow is optimized for a specific High-Performance Server (HPS), we first ran FORCE and the Nextflow port on this machine, a Linux server with an Intel Xeon Platinum 8176M CPU (56/112 cores/hyperthreads, 2.10GHz), 750GB main memory, and 98TB disc space (RAID6, ext4). Running the Nextflow workflow on this machine allows measuring the effects of our changes to the workflow and the computational overhead imposed by the more complex parallelizing strategy.

Experiment 2 (Cluster performance): To analyze the scalability of the Nextflow workflow, we ran it on a local cluster consisting of 27 homogeneous nodes, each equipped with an Intel Xeon E3-1230 V2 CPU (Quad-Core, 3.30Ghz), 16GB main memory, and three 1TB hard disks. Nodes are connected with a 1Gbit/s network link. Note that this is a somewhat outdated hardware, as clusters today typically have much faster network (e.g. 10GBit/s) and more memory per node (e.g. 128GB). Slow networks impede I/O hungry tasks, whereas small main memory reduces the degree-of-parallelism for memory-hungry tasks. Ceph was configured to use two 1TB hard drives per node as block storage, resulting in 12TB raw storage capacity on six nodes. In order to improve I/O operations, two 1Gbit/s network interfaces of the nodes are utilized to segment Ceph management and storage traffic. We ran the workflow using 1, 3, 5, 10, 14, 15, 20, and 21 compute nodes. Note that the 14 node setup offers the same number of cores and threads as the HPS.

[6] https://github.com/nextflow-io/nextflow/pull/2182
[7] https://github.com/nextflow-io/nextflow/pull/2174

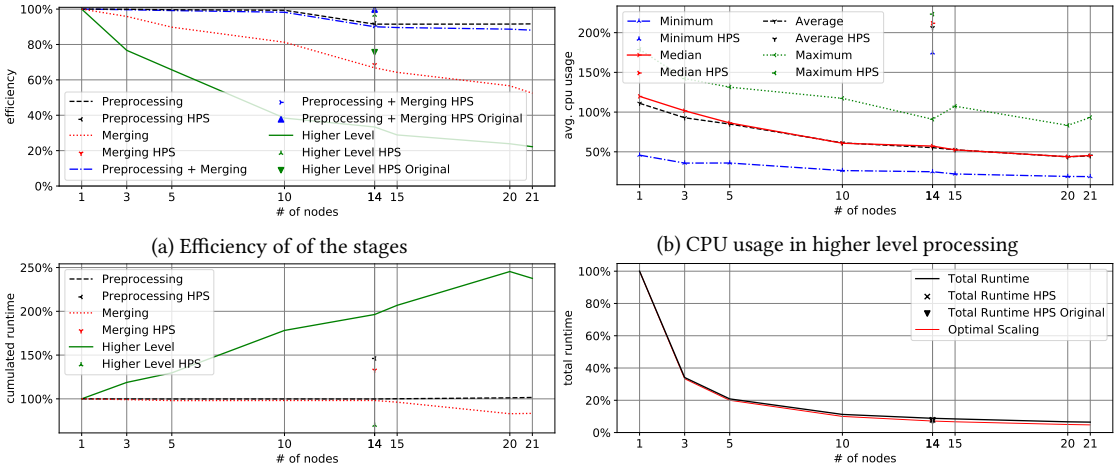
6. Results

Results are summarised in Figure 2. The X-axis in all graphics determines the number of nodes used; the runtime of the HPS setup is drawn at x=14, in addition to the runtimes of the 14 node cluster. We first describe the overall results of Experiment 1 and 2 and then provide a detailed analysis of individual workflow stages. We published all logs, additional plots, a tabular preparation of results, and the analysis script in our GitHub repository.

Experiment 1: HPS: We first run FORCE on HPS, where it finishes after 358min. Running the Nextflow workflow on the same machine requires 384min. The 7.5% increase in runtime is expected as a system like Nextflow for a single server only adds overhead without offering any performance benefits.

Experiment 2: Cluster: We can only measure the Nextflow workflow on the cluster as FORCE does not support execution on Kubernetes. Running the Nextflow workflow on a single node of the cluster took 4,883min, which decreased to 315min when using all 21 nodes (Figure 2d). This implies a reduction of a factor of 15.5, which is considerably less than the ideal factor of 21, hinting at problems in the parallelization. Using 21 nodes is thus 13.4% faster than using the HPS despite the slow network. On HPS, on the other hand, the Nextflow workflow required only 11.0% less time than on a cluster with 14 nodes (i.e., an equal number of cores), meaning that the overhead of managing tasks and files in a distributed setting is surprisingly low.

Preprocessing and merging: Preprocessing took 4,338min on one cluster node; using 21 nodes decreased runtime to 225min, i.e., a 19.2-fold improvement with an efficiency of 91.6% (Figure 2a). On HPS, it took 337min and 339min in our cluster with 14 nodes. The preprocessing is CPU bound; accordingly, the cumulative runtime is constant over all experiments (Figure 2c). The HPS shows a 46.4% higher cumulative runtime, which is likely due to a different CPU architecture. As the preprocessing



(a) Efficiency of of the stages

(b) CPU usage in higher level processing

(c) Cumulated runtime of the stages

(d) Total Runtime

Figure 2: Experiment's results: All data points are the median for the measured values over all three executions. HPS is the high-performance server with 56 cores, respectively comparable to 14 nodes with four cores. The runtime on one node serves as a baseline to calculate the relative duration for all stages, and their efficiency; efficiency is the proportion a perfect scaling would have.

scales quite favorably, we conclude that our uncoupling as described in Section 4 works well. Only in the stage's beginning, all tasks read 80 – 204MB simultaneously, which might decrease the efficiency when more nodes are used than available here.

The merging stage scales slightly inferiorly. The efficiency decreases by around 2.5% per additional node, resulting in a drop in the cumulative runtime when 20 or more nodes are used. We assume this is because the merge task is I/O-bound and thus heavily limited by the slow network. With more nodes, the load is fairly distributed, and the individual connections are relieved.

The workflow reaches an efficiency of 90.0% for the aggregated preprocessing and merging stage when using 14 nodes (56 cores). This setup is slightly slower than the FORCE workflow on HPS (same number of cores); it surpasses FORCE from 16 nodes on, finishing 24.5% earlier than FORCE on HPS when using all 21 nodes.

Higher Level Processing: The higher level processing stage scales considerably worse because it is very I/O intensive, which turns the slow network into a major bottleneck (see the decrease of average CPU usage in Figure 2b). This stage takes 319min on one node, which is only 4.7 times longer than using 21 nodes. The cumulative runtime increased significantly (Figure 2c), growing to 245.4% for 20 nodes as the I/O times rise for each task.

The I/O problems in this stage also become apparent when comparing HPS with the cluster. Using 14 nodes, the Nextflow workflow had an efficiency of 33.4% (Figure 2a). Running the Nextflow workflow on the HPS with the same number of cores, efficiency was 96.6%. The throughput increased through (a) Linux's caching capabilities combined with the large memory, and (b) the RAID6 configuration. The original setup is not able to benefit as much of these features, reaching only 75.6% efficiency on HPS, because it is not optimized for the case where the preprocessing directly runs before and data is still in the cache. Thus, FORCE optimizes the read patterns in the higher level processing. Therefore, it already reads the following image blocks while processing the current ones (see Section 2) to reduce I/O wait times, leading to less idle CPUs. The tiles, in turn, are processed in sequential order, using all available cores via shared memory parallelization, which also adds some synchronization overhead. In Nextflow, multiple tiles are processed in parallel leading to parallel reads. In the distributed setting this overloads the network switch, resulting in high I/O wait times and hence longer runtime (Figure 2c). On HPS, many of these parallel reads are avoided by exploiting the large caching capacities, leading to lower runtime for Nextflow than for FORCE.

7. Discussion and Conclusion

Overall, our experiments show that a workflow engine like Nextflow can help to make EO workflows more scalable and to ease their reuse on different infrastructures. The benefits should especially pay off when using larger clusters for larger input data sets which should be possible without any changes to the workflow. Besides scalability concerns, the initial price one has to pay for learning the specificities of a workflow engine and the peculiarities of a distributed execution is notable; this investment, on the other hand, should pay off quickly when more and more workflows are to be implemented or truly large analysis tasks, exceeding the capabilities of stand-alone

servers, are approached. In the following, we discuss a number of further and more detailed conclusions we draw from our experiments.

Scalability of stages

Overall, our Nextflow workflow scaled quite well, reaching an efficiency of 73.7%. Nextflow's distribution and parallelization techniques offer a comparable performance to manual ones, outperforming them in exceptional cases. The scalability, however, is not equal between the different stages. While preprocessing and merging scale very well, higher level processing suffers a lot from the distributed data access and slow network. This should be taken into account when groups focus their research on only one of these aspects. The preprocessing step is more and more transitioning into the hands of institutional data providers through the widespread emergence of ARD (see e.g. [29]), thus data providers would benefit largely from using a workflow management system. End users, however, increasingly start from existing ARD and thus focus on higher level analyses, which at least in our setup, did not benefit as much from the workflow porting. We are, however, confident that also this stage can be adapted better for a distributed execution, for instance, by using some compute nodes as storage nodes or by using a location-aware scheduler to reduce data movement and better explicit operation system caching. Besides, we would already expect a quite different behavior when running the workflow on a cluster with more up-to-date network bandwidth.

The scalability of other higher-level EO workflows might also differ. For example, more compute-heavy workflows with machine learning-based components (e.g. [30]) require much more computational power and thus outweigh the I/O time as the limiting factor. However, these workflows usually include a feature generation task with a similar resource requirement as the higher-level workflow used in this paper.

Developing EO workflows with Nextflow

Scientific workflow systems like Nextflow promise additional benefits besides scalability. We give a short account on our experiences during porting.

First, Nextflow achieves automatic parallelization, adequate distributed scheduling, and distributed file handling. For scheduling and parallelization, each task in Nextflow is defined with its required memory and CPU, wherein FORCE explicitly defines the level of parallelism. Not having to implement these features oneself is a considerable benefit. On the other hand, designing workflows with Nextflow requires "distributed thinking" and prevents many tricks that are possible in local machines. For smaller analysis tasks, this probably is an overhead; for larger analysis tasks, reusing such preexisting distributed software infrastructures seems inevitable.

Second, programming a workflow in Nextflow requires more code than in a Bash script. Tasks must be wrapped, and their inputs, outputs, and parameters defined. This creates boilerplate code which can make a workflow more difficult to read and debug. These effects occur especially when existing code is ported into a workflow system; they are probably less of an issue when workflows are designed ab-initio for such a distributed system.

Third, while Nextflow supports multiple resource managers, it is partially impossible to reuse the same parameters in different systems. For example, the memory usage varies on other machines even for the same task/input combination since I/O speed differs and tool-internal caching gets more heavily used. This could also happen within a heterogeneous cluster. We plan to develop automatic, accurate, and in particular input-dependent resource predictions for future work [31, 32].

Fourth, Nextflow (and Kubernetes) are under continuous development. There are a number of features one could imagine to make their application easier and their executions even more scalable. For instance, task aggregation [33] to tune the start-up/runtime ratio (see Section 4) is a known technique from high performance computing that did not yet make it into the workflow world. Handling directories could be improved to offer a more powerful file management (see Section 4). Offering a shared-memory and/or streaming interface between tasks could help to reduce intermediate I/O, although it must be carefully designed to also work smoothly in a distributed setting since it creates additional constraints.

Finally, Nextflow also offers a number of features we did not further discuss here. For instance, it is capable to resume workflow execution after a stop or crash, which avoids a lot of unnecessary computation and comes in very handy during workflow development. However, we faced problems with the resume functionality when reading input files through the GDAL API in the higher level processing, thus disabling Nextflow's resume capability. Moreover, Nextflow offers comfortable monitoring features while running a workflow and extensive logging to perform runtime profiling, analysis, and tuning. Finally, it offers the Nextflow Tower[8] for managing and administering entire repositories of workflows, which becomes important when organizations grow larger.

To compare Nextflow with other workflow engines, we also work on porting the workflow to Apache Airflow[9]. Nevertheless, while porting the workflow, Nextflow was very intuitive and helped us in understanding the workflow and tasks' dependencies due to its simple DSL.

[8] https://tower.nf/
[9] https://airflow.apache.org/

Acknowledgments

Landsat data courtesy of the U.S. Geological Survey. We thank Google for openly mirroring Landsat collection 1 on the Google Cloud Platform. We thank the two anonymous reviewers for their valuable feedback on our manuscript. This work was funded by the German Research Foundation (DFG), CRC 1404: "FONDA: Foundations of Workflows for Large-Scale Scientific Data Analysis".

References

[1] S. Cohen-Boulakia, K. Belhajjame, O. Collin, J. Chopard, C. Froidevaux, A. Gaignard, K. Hinsen, P. Larmande, Y. L. Bras, F. Lemoine, F. Mareuil, H. Ménager, C. Pradal, C. Blanchet, Scientific workflows for computational reproducibility in the life sciences: Status, challenges and opportunities, Future Generation Computer Systems 75 (2017) 284–298. doi:10.1016/j.future.2017.01.012.

[2] C. Schiefer, M. Bux, J. Brandt, C. Messerschmidt, K. Reinert, D. Beule, U. Leser, Portability of Scientific Workflows in NGS Data Analysis: A Case Study, 2020. arXiv:2006.03104.

[3] P. Di Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, C. Notredame, Nextflow enables reproducible computational workflows, Nature Biotechnology 35 (2017) 316–319. doi:10.1038/nbt.3820.

[4] M. Beauchemin, Airflow: a workflow management platform, 2015. URL: https://medium.com/airbnb-engineering/airflow-a-workflow-management-platform-46318b977fd8.

[5] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, M. Livny, Pegasus: Mapping Scientific Workflows onto the Grid, in: D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, M. Sudan, D. Terzopoulos, D. Tygar, M. Y. Vardi, G. Weikum, M. D. Dikaiakos (Eds.), Grid Computing, volume 3165, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 11–20. doi:10.1007/978-3-540-28642-4_2, series Title: Lecture Notes in Computer Science.

[6] A. Siretskiy, T. Sundqvist, M. Voznesenskiy, O. Spjuth, A quantitative assessment of the Hadoop framework for analyzing massively parallel DNA sequencing data, GigaScience 4 (2015) 26. doi:10.1186/s13742-015-0058-5.

[7] B. Balis, K. Figiela, K. Jopek, M. Malawski, M. Pawlik, Porting HPC applications to the cloud: A multi-frontal solver case study, Journal of Computational Science 18 (2017) 106–116. doi:10.1016/j.jocs.2016.09.006.

[8] J. Cała, E. Marei, Y. Xu, K. Takeda, P. Missier, Scalable and efficient whole-exome data processing using workflows on the cloud, Future Generation Computer Systems 65 (2016) 153–168. doi:10.1016/j.future.2016.01.001.

[9] J. Ossyra, A. Sedova, A. Tharrington, F. Noé, C. Clementi, J. C. Smith, Porting Adaptive Ensemble Molecular Dynamics Workflows to the Summit Supercomputer, in: M. Weiland, G. Juckeland, S. Alam, H. Jagode (Eds.), High Performance Computing, volume 11887, Springer International Publishing, Cham, 2019, pp. 397–417. doi:10.1007/978-3-030-34356-9_30, series Title: Lecture Notes in Computer Science.

[10] D. Frantz, FORCE—Landsat + Sentinel-2 Analysis Ready Data and Beyond, Remote Sensing 11 (2019) 1124. doi:10.3390/rs11091124.

[11] D. Frantz, Generation of Higher Level Earth Observation Satellite Products for Regional Environmental Monitoring, Phd dissertation, Universität Trier, 2017. doi:10.25353/ubtr-xxxx-a477-6262.

[12] P. Hostert, A. Röder, J. Hill, Coupling Spectral Unmixing and Trend Analysis for Monitoring of Long-Term Vegetation Dynamics in Mediterranean Rangelands, Remote Sensing of Environment 87 (2003) 183–197. doi:10.1016/S0034-4257(03)00145-7.

[13] P. Hostert, A. Röder, J. Hill, T. Udelhoven, G. Tsiourlis, Retrospective studies of grazing-induced land degradation: A case study in central Crete, Greece, International Journal of Remote Sensing 24 (2003) 4019–4034. doi:10.1080/0143116031000103844.

[14] D. Frantz, M. Stellmes, S. Ernst, Water vapor database for atmospheric correction of Landsat imagery, 2021. doi:10.5281/zenodo.4468701.

[15] D. Frantz, M. Stellmes, P. Hostert, A Global MODIS Water Vapor Database for the Operational Atmospheric Correction of Historic and Recent Landsat Imagery, Remote Sensing 11 (2019) 257. doi:10.3390/rs11030257.

[16] Z. Zhu, C. E. Woodcock, Object-Based Cloud and Cloud Shadow Detection in Landsat Imagery, Remote Sensing of Environment 118 (2012) 83–94. doi:10.1016/j.rse.2011.10.028.

[17] Z. Zhu, S. Wang, C. E. Woodcock, Improvement and Expansion of the Fmask Algorithm: Cloud, Cloud Shadow, and Snow Detection for Landsats 4–7, 8, and Sentinel 2 Images, Remote Sensing of Environment 159 (2015) 269–277. doi:10.1016/j.rse.2014.12.014.

[18] D. Frantz, A. Röder, T. Udelhoven, M. Schmidt, Enhancing the Detectability of Clouds and Their Shad-

ows in Multitemporal Dryland Landsat Imagery: Extending Fmask, IEEE Geoscience and Remote Sensing Letters 12 (2015) 1242–1246. doi:10.1109/lgrs.2015.2390673.

[19] D. Frantz, A. Röder, M. Stellmes, J. Hill, An Operational Radiometric Landsat Preprocessing Framework for Large-Area Time Series Applications, IEEE Transactions on Geoscience and Remote Sensing 54 (2016) 3928–3943. doi:10.1109/TGRS.2016.2530856.

[20] J. Buchner, H. Yin, D. Frantz, T. Kuemmerle, E. Askerov, T. Bakuradze, B. Bleyhl, N. Elizbarashvili, A. Komarova, K. E. Lewińska, A. Rizayeva, H. Sayadyan, B. Tan, G. Tepanosyan, N. Zazanashvili, V. C. Radeloff, Land-cover change in the Caucasus Mountains since 1987 based on the topographic correction of multi-temporal Landsat composites, Remote Sensing of Environment 248 (2020) 111967. doi:10.1016/j.rse.2020.111967.

[21] D. P. Roy, H. K. Zhang, J. Ju, J. L. Gomez-Dans, P. E. Lewis, C. B. Schaaf, Q. Sun, J. Li, H. Huang, V. Kovalskyy, A General Method to Normalize Landsat Reflectance Data to Nadir BRDF Adjusted Reflectance, Remote Sensing of Environment 176 (2016) 255–271. doi:10.1016/j.rse.2016.01.023.

[22] D. A. Roberts, J. B. Adams, M. O. Smith, Predicted distribution of visible and near-infrared radiant flux above and below a transmittant leaf, Remote Sensing of Environment 34 (1990) 1–17. doi:10.1016/0034-4257(90)90080-6.

[23] R. Bro, S. De Jong, A fast non-negativity-constrained least squares algorithm, Journal of Chemometrics 11 (1997) 393–401. doi:10.1002/(SICI)1099-128X(199709/10)11:5<393::AID-CEM483>3.0.CO;2-L.

[24] E. Vermote, C. O. Justice, F. M. Breon, Towards a Generalized Approach for Correction of the BRDF Effect in MODIS Directional Reflectances, Geoscience and Remote Sensing, IEEE Transactions on 47 (2009) 898–908. doi:10.1109/tgrs.2008.2005977.

[25] M. Schwieder, P. J. Leitão, M. M. da Cunha Bustamante, L. G. Ferreira, A. Rabe, P. Hostert, Mapping Brazilian savanna vegetation gradients with Landsat time series, International Journal of Applied Earth Observation and Geoinformation 52 (2016) 361–370. doi:10.1016/j.jag.2016.06.019.

[26] B.-G. J. Brooks, D. C. Lee, L. Y. Pomara, W. W. Hargrove, Monitoring Broadscale Vegetational Diversity and Change across North American Landscapes Using Land Surface Phenology, Forests 11 (2020). doi:10.3390/f11060606.

[27] J. N. Hird, G. Castilla, G. J. McDermid, I. T. Bueno, A Simple Transformation for Visualizing Non-seasonal Landscape Change From Dense Time Series of Satellite Data, IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing 9 (2016) 3372–3383. doi:10.1109/jstars.2015.2419594.

[28] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, C. Maltzahn, Ceph: A scalable, high-performance distributed file system, in: Proceedings of the 7th Symposium on Operating Systems Design and Implementation, OSDI '06, USENIX Association, USA, 2006, p. 307–320.

[29] J. Dwyer, D. Roy, B. Sauer, C. Jenkerson, H. Zhang, L. Lymburner, Analysis Ready Data: Enabling Analysis of the Landsat Archive, Remote Sensing 10 (2018) 1363. doi:10.3390/rs10091363.

[30] J. Rosentreter, R. Hagensieker, B. Waske, Towards large-scale mapping of local climate zones using multitemporal Sentinel 2 data and convolutional neural networks, Remote Sensing of Environment 237 (2020) 111472. doi:https://doi.org/10.1016/j.rse.2019.111472.

[31] C. Witt, M. Bux, W. Gusew, U. Leser, Predictive Performance Modeling for Distributed Computing using Black-Box Monitoring and Machine Learning, Information Systems 82 (2019) 33–52. doi:10.1016/j.is.2019.01.006.

[32] C. Witt, J. van Santen, U. Leser, Learning Low-Wastage Memory Allocations for Scientific Workflows at IceCube, in: 2019 International Conference on High Performance Computing & Simulation (HPCS), IEEE, Dublin, Ireland, 2019, pp. 233–240. doi:10.1109/HPCS48598.2019.9188126.

[33] A. E. Helal, A. M. Aji, M. L. Chu, B. M. Beckmann, W.-c. Feng, Adaptive Task Aggregation for High-Performance Sparse Solvers on GPUs, in: 2019 28th International Conference on Parallel Architectures and Compilation Techniques (PACT), IEEE, Seattle, WA, USA, 2019, pp. 324–336. doi:10.1109/PACT.2019.00033.