Hybrid Workflows for Large - Scale Scientific Applications

HYBRID WORKFLOWS FOR LARGE-SCALE SCIENTIFIC APPLICATIONS

Iacopo Colonnelli and Marco Aldinucci
University of Torino, Computer Science Dept.

Author's copy (preprint) of Iacopo Colonnelli and Marco Aldinucci, "Hybrid Workflows for Large - Scale Scientific Applications," in Sixth EAGE High Performance Computing Workshop, Milano, Italy, 2022, p. 1-5. doi: 10.3997/2214-4609.2022615029.

Introduction

Large-scale scientific applications are facing an irreversible transition from monolithic, HPC-oriented codes to modular and portable deployments of specialized (micro-)services. The reasons behind these transitions are many. Standard MPI-based distributed solvers are increasingly coupled with Deep Learning to gain efficiency in some specific simulation phases. Data analytics and visualization tools, crucial for interpretability, are increasingly moving from queue-based HPC clusters to more accessible on-demand cloud resources. Finally, the advent of specialized hardware accelerators (e.g., GPGPUs, neuromorphic processors, and quantum machines) fosters additional reengineering efforts to port each component of a complex pipeline to the best-suited architecture.

In this scenario, topology-aware Workflow Management Systems (WMSs) play a crucial role. The intrinsic modularity of workflow abstractions facilitates the design of heterogeneous applications. The explicit encoding of (true) data dependencies among workflow steps allows distributed runtime systems to extract the maximum amount of concurrency, improving performances. Topology-awareness allows an explicit mapping of workflow steps onto potentially heterogeneous locations, improving portability and reproducibility and allowing automated executions on top of hybrid architectures (e.g., cloud+HPC facilities or classical+quantum machines). Finally, topology-aware WMSs can relieve domain experts of the weight of implementing non-functional requirements like components' life-cycle orchestration, secure and efficient data transfers, fault tolerance, and cross-cluster execution of urgent workloads.

If the additional effort of mastering a feature-rich coordination language is worth it in the case of production-ready workflows, it can be undue stress when prototyping a novel research approach. However, also proof-of-concept experiments should be reproducible, and in many situations, they also need to scale. Augmenting interactive Jupyter Notebooks with distributed workflow capabilities allows domain experts to prototype and scale applications using the same technological stack. Jupyter Notebooks have a feature-rich and user-friendly web interface, which is far more accessible than the SSH-based remote shells commonly exposed by HPC facilities worldwide.

This workshop will showcase how these general methodologies can be applied to a typical geoscience simulation pipeline based on the Full Wavefront Inversion (FWI) technique. In particular, a prototypical Jupyter Notebook will be executed interactively on the cloud. Preliminary data analyses and post-processing will be executed locally on a commodity virtual machine. Conversely, the computationally demanding optimization loop will be scheduled on a remote HPC cluster to improve performance.

Hybrid workflows and StreamFlow

A *hybrid workflow* is a workflow whose steps can span multiple, heterogeneous, and independent computing infrastructures. Support for *multiple* infrastructures implies that each step can target a different *location* in charge of executing it. That location must have access to all the input data and will contain all the output data after the execution. Locations can be *heterogeneous*, either at the hardware level (e.g., containing GPUs, FPGAs, or quantum devices) or at the software level (e.g., exposing different interfaces for authentication, communication, and scheduling). Finally, direct communications and data transfers among *independent* locations may not be allowed.

A suitable hybrid workflow model should then couple a standard workflow model with the topology of available locations and mapping between steps and locations. To be consistent with actual WMSs implementations, the model must distinguish between *control* locations belonging to the WMS control plane and *processing* locations that can only execute or delegate commands to its neighbors. The presence of a channel from location l_1 to location l_2 means that location l_1 can initiate a connection to l_2, even if the connection is bidirectional. A mapping relation identifies the set of locations in charge of executing a step. Note that multiple steps mapped to the same location create a *temporal constraint*, i.e., concurrent executions can be serialized if the location does not have enough resources to process them in parallel. Conversely, a step mapped to multiple locations creates a *spatial constraint*, i.e., all locations must be available simultaneously to start the execution. Figure 1 shows an example of hybrid workflow mapped onto a hybrid cloud+HPC environment.

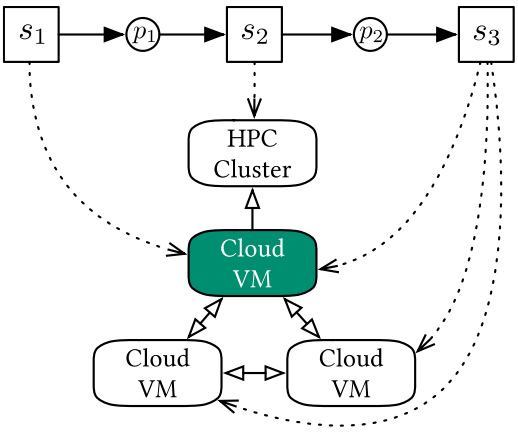

Figure 1 *Example of hybrid workflow model.*

The StreamFlow framework[1] is a runtime support for hybrid workflows written entirely in Python [1]. StreamFlow has been designed to seamlessly integrate with external coordination semantics, allowing users to augment existing workflows with hybrid capabilities. In particular, it is fully compliant with the Common Workflow Language (CWL) open standard [2], also supporting scatter/gather data parallel patterns. Steps can be mapped onto several locations: Docker or Singularity containers, complex microservice-based applications described with Docker Compose files or Helm charts, SSH-equipped cloud VMs or bare metal nodes, and HPC workload managers (Slurm or PBS). Plus, its plugin mechanism makes it easy to add new location types or to replace some other features with custom implementations, e.g., the scheduling policy, the fault-tolerance strategy, or the data management layer.

Literate workflows and Jupyter Workflow

Despite all the advantages of scientific WMSs in modularity, portability, and reproducibility, domain experts often prefer to stick with standard, general-purpose languages to develop and publish their experiments. Some apparent reasons behind this choice are the additional effort required to learn a new framework, the increased difficulty in maintaining coherence between host and coordination logic, and the lack of a de-facto standard WMS that everyone should know and use.

The *literate workflows* paradigm derives from the concept of literate computing [3]. It interleaves host and coordination logic in the same document (a computational notebook) but at the same time keeps them separated. A computational notebook is a list of code cells executed sequentially in a given order. The idea is to treat each cell as a workflow step, using the related metadata to express input and output dependencies, locations, mapping relations, data-parallel patterns and other properties. A workflow graph can be extracted from this representation, and independent steps can be executed concurrently in

[1] https://streamflow.di.unito.it

different locations, modelling *hybrid literate workflows*. It is possible to show [4] that that if an associative operator can properly reconcile conflicting identifiers (i.e., clashes due to Bernstein's output dependencies), then a strategy exists to obtain a sequentially equivalent workflow graph.

Jupyter Workflow[2] extends the IPython software stack to support hybrid literate workflows [4]. Its logical architecture consists of three main components. A JSON-based coordination metadata format allows users to model cell configurations and location topologies standardized through the Notebook's metadata. A dependency resolver automatically identifies the input dependencies of each cell by inspecting the code's Abstract Syntax Tree (AST). Finally, a Jupyter stack extension hides workflow metadata behind a Graphical User Interface (GUI), communicates them to the Jupyter kernel and relies on StreamFlow to build and orchestrate the hybrid workflow graph.

Evaluation

We tested the capabilities of Jupyter Workflow on a typical geoscience simulation pipeline based on the Full Wavefront Inversion (FWI) technique [5]. Applying the FWI, it is possible to create an image of the subsurface from velocity recorded data, solving the so-called *seismic inversion problem*. The FWI can be expressed as follows:

$$\min_m \Phi_s(m) = \frac{1}{2} \|P_r A(m)^{-1} P_s^T q_s - d\|_2^2$$

where m is the squared slowness of the wave, P_r is the sampling operator at the receiver location, P_s^T is the injection operator at the source location, $A(m)$ is an operator that represents the discretized wave equation matrix, $u = A(m)^{-1} P_s^T q_s$ is the discrete synthetic pressure wavefield, q_s is the pressure source, and d is the measured data. This problem can be solved using a gradient-based method:

$$\nabla \Phi_s(m) = \sum_{t=1}^{n_t} u[t]\, v_{tt}[t] = J^T \delta d_s$$

where n_t is the number of computational time steps, $\delta d_s = (P_r u - d)$ is the residual between measured and modelled data, J is the Jacobian operator, and v_{tt} is the second-order time derivative of the adjoint wavefield, such that $A^T(m) v = P_r^T \delta d$.

We started from a Jupyter Notebook[3] that uses the Devito library [6] to solve the FWI for a circular 2D model domain. *Devito* is a performance-oriented package implementing optimized stencil computation through just-in-time (JIT) compilation. It can benefit from the presence of physical hardware (exploiting its capabilities to fine-tune kernels), parallelize computation with MPI+OpenMP, and rely on hardware accelerators. In this workshop, we concentrate on CPU-based computations. Firstly, we ran the Jupyter Workflow notebook on a cloud VM with 8 cores and 32 GB RAM; secondly, we offloaded heavy computations on a bare metal node with a more performant Intel Xeon Gold 6230 (40 cores, 2.10 GHz) and 1.47TB RAM. Except for the gradient computation cell, all the notebook cells execute immediately on the local VM. Therefore, we concentrate on gradient computation performances.

As first step we tested the multicore scalability within a single node; we configured Devito to compile OpenMP-based versions of the kernels. By properly setting the `OMP_NUM_THREADS` variable, we studied the scalability up the maximum core count on the VM and the bare metal node. Unexpectedly, the time-to-solution remained almost constant, independent of the number of involved threads. Looking at the generated C code, OpenMP is only used for parallelizing (via OMP SIMD directive) of a tiny portion of the code and the execution time, resulting in little to no speedup (in conformity with Amdahl's Law).

[2] https://jupyter-workflow.di.unito.it

[3] https://github.com/devitocodes/FWI_lectures/blob/main/lecture11/L11_numerical_implementations_of_fwi.ipynb

As a second step, we studies te overhead intrioduced by the VM against using a bare metal node. For this setting, we kept `OMP_NUM_THREADS` equal to 1. We scaled the problem size by doubling nt at each iteration, moving from 21 to 336 (16x). The result is an almost 2x speed increase on the bare metal machine, which is higher than expected and the motivation for which is yet to be analyzed (the ratio between vcpu and cores is 1:1). Note that Devito kernels are JIT-recompiled on the remote location, potentially with a different configuration, to maximize performance on available resources. Obtaining this behavior without changing the business code is a valuable feature per se. However, we are still using a single core.

Finally, we further parallelized the code according to the methodology described in [7] by turning the gradient computation loop into a map-reduce computation: first compute all the residuals in parallel, then reduce them by sum to compute the gradient. The benefit is that multiple residuals can be computed in parallel by multiple cores or even multiple nodes, reaching better scalability. However, the main drawback is that the problem becomes I/O bound, as the residual data structures are large (tens of GBs) and they should be moved across locations. For this, minimizing data movement and adopting high-speed communication channels is crucial for performance, highlighting the HPC nature of the problem. A promising strategy, which will be investigated in the future, is to work at the file-system level to enable in-memory streaming data movements (using efficient *burst buffers* for swapping) without changing the application code. This mechanism allows starting the reduce phase as soon as data structures become available and avoid involving the shared file system.

Acknowledgements

This article describes work undertaken in the context of the ACROSS project[4], ‘‘*HPC Big Data Artificial Intelligence Cross Stack Platform Towards Exascale*’’, and the EUPEX project[5], “*European Pilot for Exascale*”, which received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No. 955648 and 101033975, respectively. We also want to give a special thanks to Nicola Bienati (Eni S.p.A.) for providing us the evaluation Notebook and for the fundamental advice on how to approach scalability of a seismic inversion problem.

References

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

[2] M. R. Crusoe, S. Abeln, A. Iosup, P. Amstutz, J. Chilton, N. Tijanic, H. Ménager, S. Soiland-Reyes, B. Gavrilovic, C. A. Goble and The CWL Community, "Methods Included: Standardizing Computational Reuse and Portability with the Common Workflow Language," *Communications of the ACM*, vol. 65, no. 6, pp. 54-63, 2022.

[3] K. J. Millman and F. Pérez, "Developing open-source scientific practice," in *Implementing Reproducible Research*, Chapman and Hall/CRC, 2014, pp. 149-183.

[4] I. Colonnelli, M. Aldinucci, B. Cantalupo, L. Padovani, S. Rabellino, C. Spampinato, R. Morelli, R. Di Carlo, N. Magini and C. Cavazzoni, "Distributed workflows with Jupyter," *Future Generation Computer Systems*, vol. 128, pp. 282-298, 2022.

[5] A. Fichtner, Full Seismic Waveform Modelling and Inversion, Springer Berlin, Heidelberg, 2011.

[6] F. Luporini, M. Louboutin, M. Lange, N. Kukreja, P. Witte, J. Hückelheim, C. Yount, P. H. J. Kelly, F. J. Herrmann and G. J. Gorman, "Architecture and Performance of Devito, a System for Automated Stencil Computation," *ACM Transactions on Mathematical Software*, vol. 46, no. 1, 2020.

[4] https://acrossproject.eu
[5] https://eupex.eu

[7] M. Aldinucci, V. Cesare, I. Colonnelli, A. R. Martinelli, G. Mittone, B. Cantalupo, C. Cavazzoni and M. Drocco, "Practical parallelization of scientific applications with OpenMP, OpenACC and MPI," *Journal of Parallel and Distributed Computing,* vol. 157, pp. 13-29, 2021.