HPC Application Cloudification: The StreamFlow Toolkit

Iacopo Colonnelli
Computer Science Department, University of Torino, Italy

Barbara Cantalupo
Computer Science Department, University of Torino, Italy

Roberto Esposito
Computer Science Department, University of Torino, Italy

Matteo Pennisi
Electrical Engineering Department, University of Catania, Italy

Concetto Spampinato
Electrical Engineering Department, University of Catania, Italy

Marco Aldinucci
Department of Computer Science, University of Pisa, Italy

Abstract

Finding an effective way to improve accessibility to High-Performance Computing facilities, still anchored to SSH-based remote shells and queue-based job submission mechanisms, is an open problem in computer science.

This work advocates a cloudification of HPC applications through a cluster-as-accelerator pattern, where computationally demanding portions of the main execution flow hosted on a Cloud Finding an effective way to improve accessibility to High-Performance Computing facilities, still anchored to SSH-based remote shells and queue-based job submission mechanisms, is an open problem in computer science.

This work advocates a cloudification of HPC applications through a cluster-as-accelerator pattern, where computationally demanding portions of the main execution flow hosted on a Cloud infrastructure can be offloaded to HPC environments to speed them up. We introduce StreamFlow, a novel Workflow Management System that supports such a design pattern and makes it possible to run the steps of a standard workflow model on independent processing elements with no shared storage.

We validated the proposed approach's effectiveness on the CLAIRE COVID-19 universal pipeline, i.e. a reproducible workflow capable of automating the comparison of (possibly all) state-of-the-art pipelines for the diagnosis of COVID-19 interstitial pneumonia from CT scans images based on Deep Neural Networks (DNNs).

2012 ACM Subject Classification Computer systems organization → Cloud computing; Computing methodologies → Distributed computing methodologies

Keywords and phrases cloud computing, distributed computing, high-performance computing, streamflow, workflow management systems

Digital Object Identifier 10.4230/OASIcs.PARMA-DITAM.2021.5

Category Invited Paper

Supplementary Material *Software (Stable)*: `https://github.com/alpha-unito/streamflow`
archived at `swh:1:dir:34c00b970f3326937c7b1cb6467a4a2c4f5dbdec`
Other (StreamFlow website): `https://streamflow.di.unito.it`

Funding This work has been undertaken in the context of the DeepHealth project, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 825111. This work has been partially supported by the HPC4AI project funded by Regione Piemonte (POR FESR 2014-20 - INFRA-P).

© Iacopo Colonnelli, Barbara Cantalupo, Roberto Esposito, Matteo Pennisi, Concetto Spampinato, and Marco Aldinucci;
licensed under Creative Commons License CC-BY 4.0

12th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and 10th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms (PARMA-DITAM 2021).
Editors: João Bispo, Stefano Cherubin, and José Flich; Article No. 5; pp. 5:1–5:13
OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Acknowledgements We want to thank Emanuela Girardi and Gianluca Bontempi who are coordinating the CLAIRE task force on COVID-19 for their support, and the group of volunteer researchers who contributed to the development of CLAIRE COVID-19 universal pipeline, they are: Marco Calandri and Piero Fariselli (Radiomics & medical science, University of Torino, Italy); Marco Grangetto, Enzo Tartaglione (Digital image processing Lab, University of Torino, Italy); Simone Palazzo, Isaak Kavasidis (PeRCeiVe Lab, University of Catania, Italy); Bogdan Ionescu, Gabriel Constantin (Multimedia Lab @ CAMPUS Research Institute, University Politechnica of Bucharest, Romania); Miquel Perello Nieto (Computer Science, University of Bristol, UK); Inês Domingues (School of Sciences University of Porto, Portugal).

1 Introduction

If the technical barriers to Cloud-based infrastructures lowered substantially with the advent of the *-as-a-Service model, most High-Performance Computing (HPC) facilities worldwide are still anchored to SSH-based remote shells and queue-based job submission mechanisms.

Finding an effective way to improve accessibility to this class of computing resources is still an open problem in computer science. Indeed, the multiple layers of virtualisation that characterise modern cloud architectures introduce significant processing overheads and make it impossible to apply adaptive fine-tuning techniques based upon the underlying hardware technologies, making them incompatible with performance-critical HPC applications. On the other hand, HPC centres are not designed for general purpose applications. Only scalable and computationally demanding programs can effectively benefit from the massive amount of processing elements and the low-latency network interconnections that characterise HPC facilities, justifying the high development cost of HPC-enabled applications. Moreover, some seemingly trivial applications are not supported in HPC environments, e.g. exposing a public web interface for data visualisation in an air-gapped worker node.

In this work, we promote a *cluster-as-accelerator* design pattern, in which cluster nodes act as general interpreters of user-defined tasks sent by a general-purpose host executor residing on a Cloud infrastructure, as a way to offload computation to HPC facilities in an intuitive way. Moreover, we propose *hybrid workflows*, i.e. workflows whose steps can be scheduled on independent and potentially not intercommunicating execution environments, as a programming paradigm to express such design pattern.

The StreamFlow toolkit [8], a Workflow Management System (WMS) supporting workflow executions on hybrid Cloud-HPC infrastructures, is then used to evaluate the effectiveness of the proposed approach on a real application, a Deep Neural Network (DNN) training pipeline for COVID-19 diagnosis from Computed Tomography (CT) scans images.

In more detail, Section 2 gives a general background on workflow models and discusses the state of the art, while Section 3 introduces the proposed approaches. Section 4 describes the StreamFlow toolkit, which has been used to implement the COVID-related use case described in Section 5. Finally, Section 6 summarises conclusions and future works.

2 Background and related work

A workflow is commonly represented as an acyclic digraph $G = (N, E)$, where nodes refer to different portions of a complex program and edges encode *dependency relations* between nodes, i.e. a direct edge connecting a node m to a node n means that n must wait for m to complete before starting its computation.

When expressing a program in terms of a workflow model, it is necessary to distinguish between two different classes of semantics [16]:

- The *host semantics* defining the subprograms in workflow nodes, usually expressed in a general-purpose programming language such as C++, Java or Python;
- The *coordination semantics* defining the interactions between such nodes, i.e. expressing the workflow model itself.

Tools in charge of exposing coordination semantics to the users and orchestrating workflow executions are known as Workflow Management Systems (WMSs).

Given their extreme generality, workflow models represent a powerful abstraction for designing scientific applications and executing them on a diverse set of environments, ranging from the practitioners' desktop machines to entire HPC centres. In this vision, WMSs act as an interface between the domain specialists and the computing infrastructure.

The WMS landscape is very variegated, as it embraces a broad range of very diverse scientific domains. Nevertheless, when considering the coordination semantics exposed to the workflow designers, two main classes of tools can be identified: high-level products implementing a strict separation of concerns between workflow description and application code, and low-level distributed libraries directly intermingling workflow definition with business logic.

In the first category, many tools provide a simplified Domain Specific Language (DSL) for coordination. Some WMSs adopt a Unix-style approach to define workflows in a technology-neutral way, using a syntax similar to Make [2, 14], while others provide a dataflow programming model to express parallelism easily [11, 22]. Coordination DSLs are quite flexible, but they introduce a different formalism that users must learn. For this reason, some solutions prefer to hide or replace coordination languages with a higher level Graphical User Interface (GUI), trading off complexity against flexibility [1, 17, 7].

Since product-specific DSLs and GUIs tightly couple workflow models to a single WMS, limiting portability and reusability, there are also efforts in defining vendor-agnostic coordination languages or standards. The Common Workflow Language (CWL)[1] [6] is an open standard for describing analysis workflows, following a declarative JSON or YAML syntax. Many products offer support for CWL, either alongside a proprietary coordination DSL [15] or as a higher-level semantics on top of low-level APIs [21].

A strict separation between host code and coordination logics offers a considerable abstraction level, promoting ease of understanding, portability and reproducibility across diverse execution infrastructures, and code reusability for similar purposes. Nevertheless, the significant overhead introduced by this separation of concerns makes these approaches suitable only for coarse-grained workflows, where each step performs a considerable amount of computation.

An alternative to complex and feature-rich WMSs, which puts performances first, is represented by low-level distributed libraries [23, 20, 19], which directly expose coordination programming models in the host code. Such libraries commonly allow for the execution of many interdependent tasks on distributed architectures, supporting scenarios with low-latency or high-throughput requirements. Despite being very efficient in terms of performances, these libraries are hard to use for domain experts without programming experience.

When considering targeted execution environments, both categories of solutions come with some limitations that prevent full support for hybrid architectures, i.e. mixed Cloud-HPC infrastructures. High-level products usually come with a set of pluggable connectors

[1] `https://www.commonwl.org`

targeting a broad range of infrastructures, e.g. public cloud services, batch schedulers (e.g. HTCondor, PBS, SLURM) and Kubernetes clusters. Nevertheless, in most competing approaches different steps of the same workflow cannot be managed by different connectors, forcing users to stick with the same infrastructure for the whole execution flow.

On the other side, low-level distributed libraries commonly require the presence of a shared file-system accessible by all worker nodes in the cluster (e.g. LUSTRE or HDFS), which is hardly the case in hybrid Cloud-HPC settings. Moreover, inter-node communication protocols require a bidirectional connection between the controller and the worker agents, which is not compliant with air-gapped computing nodes that usually characterise HPC facilities.

Even if some tools [10, 13] offer support for automatic data transfers among worker nodes, therefore being compatible with hybrid architectures, they rely on specific transfer protocols (e.g. GridFTP, SRM or Amazon S3) or delegate workflow management to an external batch scheduler such as HTCondor, actually constraining the set of supported configurations.

3 Hybrid workflows

The workflow abstraction has already been explored for offloading computation to HPC facilities in a transparent way. Indeed, as discussed in Section 2, many of the existing WMSs come with a diverse set of *connectors*, some of them addressing cloud environments and some others more HPC-oriented. Nevertheless, a far smaller percentage can deal with hybrid cloud/HPC scenarios for executing a single workflow.

Unfortunately, HPC facilities are not well suited for every kind of application. When executing complex workflows, it is common to have computation-intensive and highly parallelisable steps alternate with sequential or non-compute-bound operations. When scheduling an application of this kind to an HPC centre, only a subset of workflow steps will effectively take advantage of the available computing power, resulting in a low cost-benefit ratio. Moreover, some operations are not supported in HPC facilities, e.g. exposing a web interface for data visualisation in an air-gapped data centre.

In this vision, a WMS capable of dealing with *hybrid workflows*, i.e. able to schedule and coordinate different steps of a workflow on different execution environments [9], represents a crucial step towards a standard task-based interface to distributed computing. Indeed, the possibility to assign each portion of a complex application to the computing infrastructure that best suits its requirements strongly reduces the necessary tradeoffs in relying on such high-level abstraction, both in terms of performances and costs.

A fundamental step to realising a hybrid WMS is to waive the requirement for any shared data access space among all the executors, which is instead a constraint imposed by a broad range of WMSs on the market. Indeed, it is hardly the case that an HPC centre and a Cloud infrastructure can share a common file-system. This requirement can be removed by letting the runtime system automatically handle data movements among different executors whenever needed, keeping the only constraint for the WMS management infrastructure to be able to reach the whole execution environment.

This strategy enables data movements between every pair of resources, and guarantees that each of them requires at most two transfer operations: one from the source to the management infrastructure and one from the management infrastructure to the destination. Nevertheless, a two-step copy can represent an unsustainable overhead when dealing with massive amounts of data, a common scenario in modern scientific pipelines.

Making the WMS aware of a workflow's hybrid nature and letting it autonomously manage data movements are crucial aspects for performance optimisation in such scenarios. First of all, scheduling policies privileging data locality can minimise the number of required

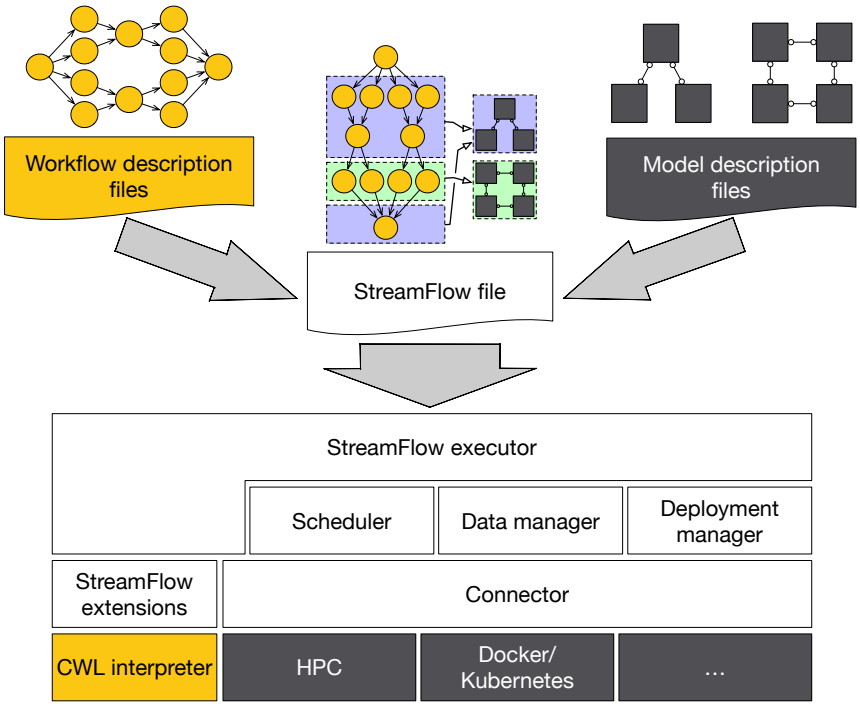

Figure 1 StreamFlow toolkit's logical stack. The yellow area is related to the definition of the workflow's dependency graph, the dark grey area refers to the execution environments, and the white portions are directly part of StreamFlow codebase.

data transfer operations, moving computation near data whenever possible. Moreover, if a WMS is aware of the underlying infrastructure topology, it can use the two-step copy strategy only as a last resort, privileging direct communication channels whenever available.

The hybrid workflows paradigm allows software architects to adopt a *cluster-as-accelerator* design pattern [12], in which cluster nodes act as general interpreters of user-defined tasks sent by a host executor. A similar pattern has been proven very effective to offload computationally heavy operations to dedicated hardware. For example, cryptographic accelerators have long been used for offloading security-related computations from the CPU, while GPGPUs programming models adopt an accelerator pattern to cooperate with the host execution flow.

In this sense, we envision hybrid workflows as a way to enable *cloudified* HPC applications, where domain experts can interact with the host execution flow through the user-friendly *-as-a-Service paradigm, but computationally demanding steps can be easily offloaded to data centers by means of high-level coordination primitives.

4 The StreamFlow toolkit

The StreamFlow[2] tookit [8], whose logical stack is depicted in Figure 1, has been specifically developed to orchestrate hybrid workflows on top of heterogeneous and geographically distributed architectures.

Written in Python 3, it can seamlessly integrate with the CWL coordination standard [6] for expressing workflow models. Alongside, one or more execution environments can be described in well-known formats, e.g. Helm charts for Kubernetes deployments or Slurm

[2] `https://streamflow.di.unito.it/`

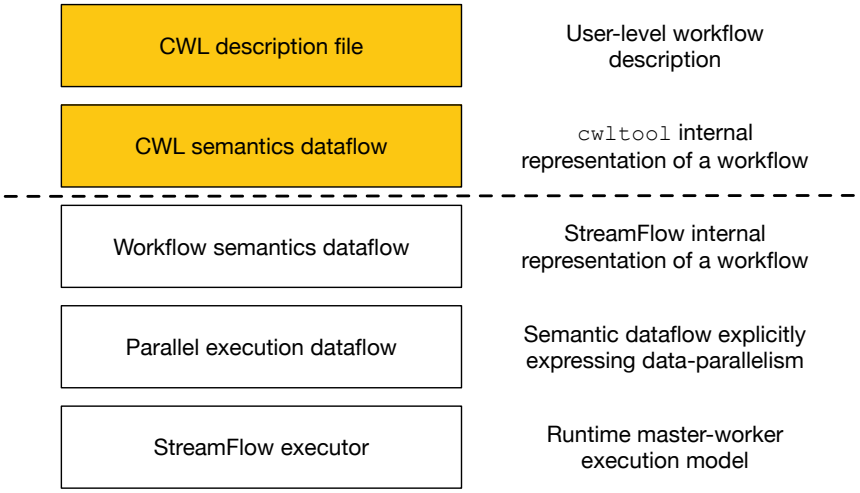

Figure 2 StreamFlow toolkit's layered dataflow model. Yellow blocks refer to the CWL runtime library's workflow representations, called cwltool, while the white ones are internal representations adopted by the different layers of the StreamFlow toolkit.

files for queue jobs. A `streamflow.yml` file, the entry point of a StreamFlow run, is then in charge of relating each workflow step with the best suitable execution environment, actually plugging the hybrid layer in the workflow design process.

CWL semantics can be used to describe a workflow through a declarative JSON or YAML syntax, written in one or more files with `.cwl` extension. Plus, an additional configuration file contains a list of input parameters to initialise a workflow execution. The CWL reference implementation, called `cwltool`, is in charge of translating these declarative semantics into an executable workflow model, such that the runtime layer can efficiently execute independent steps concurrently.

A commonly adopted executable representation of a workflow is the *macro dataflow graph* [4]. Each node of this graph can be represented as a tuple $\langle c_k, In(c_k), Out(c_k)\rangle$, where:

- c_k is a command encoding a coarse-grained computation;
- $In(c_k) = \{i_{kj} : j \in [1, m]\}$ is the set of *input ports*, i.e. the input dependencies of c_k;
- $Out(c_k) = \{o_{kj} : j \in [1, n]\}$ is the set of *output ports*, i.e. the values returned by c_k.

Each edge $\langle o_{kj}, i_{lh}\rangle$ of the graph encodes a *dependency relation* going from node k to node l, meaning that node l will receive a token on its input port i_{lh} from the output port o_{kj} of the node k. When a node receives a token on each of its input ports, it enters the *fireable* state and can be scheduled for execution. At any given time, all fireable nodes can be concurrently executed by the runtime layer, provided that enough compute units are available.

The `cwltool` library natively translates the CWL semantics in a low-level macro dataflow graph, and it also implements a multi-threaded runtime support. Nevertheless, even if CWL is the primary coordination language in StreamFlow, the integration with additional workflow design tools and formats is in plans, making it worthwhile to avoid too tight coupling between CWL logics and the StreamFlow runtime.

For this reason, the StreamFlow toolkit adopts a *layered dataflow model* [18], as depicted in Figure 2. As a first step, StreamFlow translates the CWL dataflow semantics into an internal workflow representation, explicitly modelling a macro dataflow graph. It is worth noting that this representation supports much broader semantics, including loops, stream-based input ports and from-any activation policies.

When dealing with explicit parallel semantics, whether they are data-parallel constructs like scatter/gather or stream-parallel patterns like pipeline executions, the same node of a dataflow graph can be executed multiple times. Therefore, the runtime support needs a lower, parallelism-aware layer, capable of representing each workflow step as the set of its execution units. In StreamFlow, such execution units are called *jobs* and are the only entities directly visible to the underlying runtime components for scheduling, execution and fault tolerance purposes.

Another distinctive feature of the StreamFlow WMS is the possibility to manage complex, multi-agent execution environments, ensuring the *co-allocation* of multiple heterogeneous processing elements to execute a single workflow step. The main advantage is introducing a unique interface to a diverse ecosystem of distributed applications, ranging from MPI clusters running on HPC facilities to microservices architectures deployed on Kubernetes.

To provide enough flexibility, StreamFlow adopts a three-layered hierarchical representation of execution environments:

- A `model` is an entire multi-agent infrastructure and constitutes the *unit of deployment*, i.e. all its components are always co-allocated when executing a step;
- A `service` is a single agent in a model and constitutes the *unit of binding*, i.e. each step of a workflow can be offloaded to a single service for execution;
- A `resource` is a single instance of a potentially replicated service and constitutes the *unit of scheduling*, i.e. each step of a workflow is offloaded to a configurable number of service resources to be processed.

Each model is deployed and managed independently of the others by a dedicated `Connector` implementation, which acts as a proxy for an external orchestration library. All `Connector` classes inherit from a unique base interface, so that support for different execution environments can be added to the codebase by merely developing a new implementation and plugging it in the StreamFlow runtime. In particular, the `DeploymentManager` class has the role of invoking the appropriate `Connector` implementation to create and destroy models whenever needed.

When a step becomes fireable, and the corresponding model has been successfully deployed, a `Scheduler` class is in charge of selecting the best resource to execute it. Indeed, even if only a single target service can be specified for each task, multiple replicas of the same service could exist at the same time and, if the underlying orchestrator provides auto-scaling features, their number could also change in time. Therefore, the `Scheduler` class relies on the appropriate `Connector` to extract the list of compatible resources for a given step.

Then, a scheduling policy is required to choose the best one. Given the very complex nature of hybrid workflows, a universally best scheduling strategy hardly exists. Indeed, many different factors (e.g. computing power, data locality, load balancing) can affect the overall execution time. A `Policy` interface has been introduced to allow for pluggable scheduling strategies, with a default implementation trying to minimise the data movement overhead by privileging data locality aspects.

Finally, the `DataManager` class must ensure that each service can access all its input dependencies and perform data transfers whenever necessary. As discussed in Section 3, it is always possible to move data between resources with a two-step copy operation involving the StreamFlow management infrastructure. In particular, StreamFlow always relies on this strategy for inter-model data transfers. Conversely, intra-model copies are performed by the `copy` method of the corresponding `Connector`, which is aware of the infrastructure topology and can open direct connections between resources whenever possible.

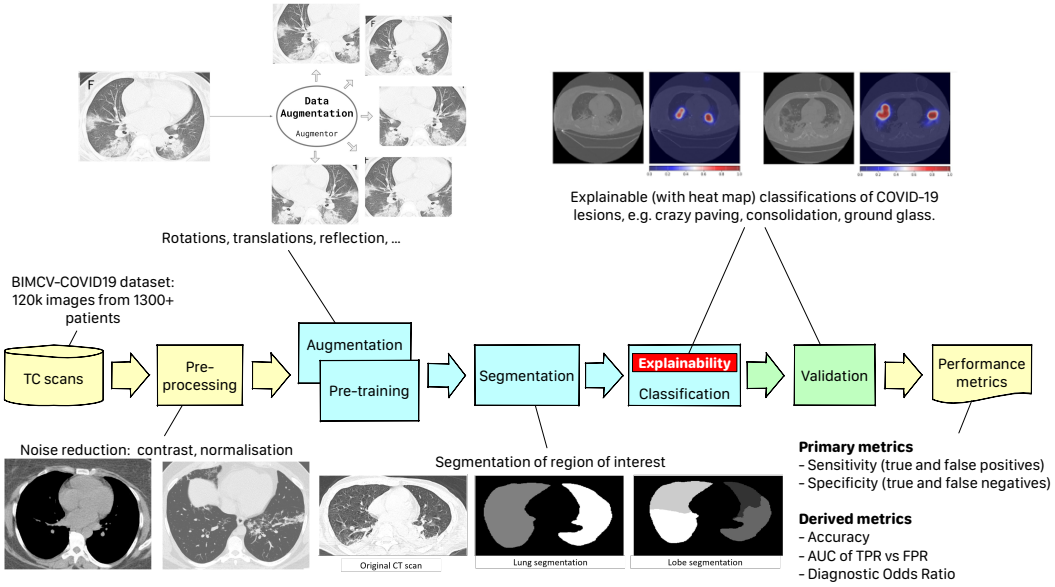

Figure 3 The CLAIRE COVID-19 universal pipeline.

It is also worth noting that all communications and data transfer operations are started and managed by the StreamFlow controller, removing the need for a bidirectional channel between the management infrastructure and the target resources. Therefore, tasks can also be offloaded to HPC infrastructures with air-gapped worker nodes, since StreamFlow directly interacts only with the frontend layer.

Moreover, StreamFlow does not need any specific package or library to be installed on the target resources, other than the software dependencies required by the host application. Therefore, virtually any target infrastructure reachable by a practitioner can serve as a target model, as long as a compatible `Connector` implementation is available.

5 Use case: the CLAIRE COVID-19 universal pipeline

To demonstrate how StreamFlow can help bridge HCP and AI workloads, we present the *CLAIRE COVID-19 universal pipeline* developed by the task force on AI & COVID-19 during the first COVID-19 outbreak to study AI-assisted diagnosis of interstitial pneumonia.

COVID-19 infection caused by the SARS-CoV-2 pathogen is a catastrophic pandemic outbreak worldwide with an exponential increase in confirmed cases and, unfortunately, deaths. When the pandemic broke out, among the initiatives aimed at improving the knowledge of the virus, containing its diffusion, and limiting its effects, the Confederation of Laboratories for Artificial Intelligence Research in Europe (CLAIRE)[3] task force on AI & COVID-19 supported the set up of a novel European group to study the diagnosis of COVID-19 pneumonia assisted by Artificial Intelligence (AI). The group was composed of fifteen researchers in complementary disciplines (Radiomics, AI, and HPC) led by Prof. Marco Aldinucci [3].

At the start of the pandemic, several studies outlined the effectiveness of radiology imaging for COVID-19 diagnosis through chest X-Ray and mainly Computed Tomography (CT), given the pulmonary involvement in subjects affected by the infection. Considering

[3] `https://https://claire-ai.org/`

the extension of the infection and the number of cases that daily emerged worldwide, the need for fast, robust, and medically sustainable diagnosis appeared fundamental. Applying artificial intelligence to radiology images to make the whole diagnosis process automatic, while reducing the efforts required by radiologists for visual inspection was relatively straight.

Even if X-Ray represents a cheaper and most effective solution for large scale screening, its low resolution led AI models to show lower accuracy than those obtained with CT data. Therefore, CT scan has become the gold standard for investigation on lung diseases. Several research groups worldwide began to develop deep-learning models for the diagnosis of COVID-19, mainly in the form of deep Convolutional Neural Networks (CNN), applying lung disease analysis from CT scans images.

As soon as we started analysing all the solution proposed, it was evident that it is impossible to select the most promising ones, due to the use of different architectures, pipelines and datasets. So, we started working on defining a reproducible workflow capable of automating the comparison of state-of-the-art deep learning models to diagnose COVID-19.

The workflow subsequently evolved towards the CLAIRE COVID-19 universal pipeline (Figure 3). This pipeline can reproduce different state-of-the-art AI models existing in the literature for the analysis of medical images. They include the pipeline for the diagnosis of COVID-19 interstitial pneumonia and other diseases. The pipeline is designed to compare the different training algorithms and therefore to define a baseline for these techniques allowing the community to quantitatively measure the progress of AI in the diagnosis of COVID-19 and similar diseases.

The universal pipeline comprises two initial steps: *Image Preprocessing* and *Augmentation*, where standard techniques for cleaning and generating variants of training images are applied. The final stages are also typical pipeline components implementing *Validation* of results and *Performance Metrics* collection.

The core steps are DNN-based. They are *Pre-training*, *Segmentation* and *Classification*. Pre-training is an unsupervised learning step and aims to generate a first set of weights for the next two steps, typically based on supervised learning. The segmentation step isolates the region of interest (e.g. lung from other tissues), and the classification step labels each image with a class identified with a kind of lesion that is typical of the disease. Each of the steps can be implemented using different DNNs, generating different variants of the pipeline. For each of these stages we selected the best DNNs that have been experimented in literature, together with a systematic exploration of networks hyperparameters, allowing a deeper search for the best model. As it can be deduced from Figure 4, the resulting number of the CLAIRE COVID-19 pipelines variants is 990.

Theoretically, the universal pipeline can reproduce the training of all the best existing and forthcoming models to diagnose pneumonia and compare their performance. Moving from theory to practice requires two non-trivial ingredients: a supercomputer of adequate computational power equipped with many latest generation GPUs and a mechanism capable of unifying and automating the execution of all variants of the workflow on a supercomputer.

To set up experiments on the pipeline, we chose the biggest dataset publicly available related to COVID-19's pathology course, i.e. BIMCV-COVID19[4], with more than 120k images from 1300 patients. Supposing to train each pre-trained model for 20 epochs on such dataset, a single variant of the pipeline takes over 15 hours on a single NVidia V100 GPU, one of the most powerful accelerators in the market with more than 5000 CUDA cores.

[4] `https://bimcv.cipf.es/bimcv-projects/bimcv-covid19/`

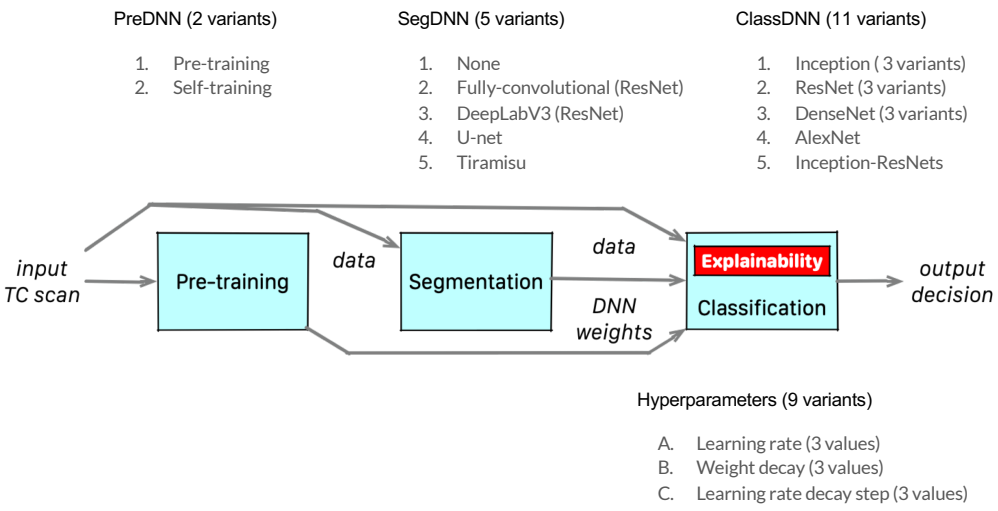

Figure 4 Core components of the CLAIRE COVID-19 universal pipeline and their variants.

Therefore, exploring all the 990 pipeline variants would take over two years on the most powerful GPU currently available and it should also be considered that tuning models for each variant would need more than one execution, too.

Fortunately, since the universal pipeline has an embarrassingly parallel structure (Figure 5), using a supercomputer could reduce the execution time down to just one day. In the best case, running all the variants concurrently on 990 different V100 GPUs only takes 15 hours of wall-clock time. Nevertheless, post-training steps like performance metrics extraction and comparison are better suited for a Cloud infrastructure, as they do not require much computing power and can significantly benefit from web-based visualisation tools. Therefore, the optimal execution of the pipeline advocates a cluster-as-accelerator design pattern.

The bridging of AI and HPC execution models has been solved by managing the universal pipeline with StreamFlow. The pipeline is naturally modelled as a hybrid workflow, offloading the training portions to an HPC facility and collecting the resulting networks on the host execution flow for visualisation purposes. As an interface towards Cloud-HPC infrastructures, StreamFlow automatically manages data movements and remote step execution, providing fault tolerance mechanisms such as checkpointing of intermediate results and replay-based recovery.

At the time of writing, the experiments have begun, and we see the first encouraging results. We performed the analysis of about 1% of the variants (10 of 990) on the High-Performance Computing for Artificial Intelligence (HPC4AI) at the University of Torino, a multi-tenant hybrid Cloud-HPC system with 80 cores and 4 GPUs per node (T4 or V100-SMX2) [5]. Results show that the CLAIRE COVID-19 universal pipeline can generate models with excellent accuracy in classifying typical interstitial pneumonia lesions due to COVID-19, with sensitivity and specificity metrics over 90% in the best cases.

6 Conclusion and future work

HPC is an enabling platform for scientific computing and Artificial Intelligence and a fundamental tool for high impact research, such as AI-assisted analysis of medical images, personalised medicine, seismic resiliency, and the green new deal. HPC and Cloud computing

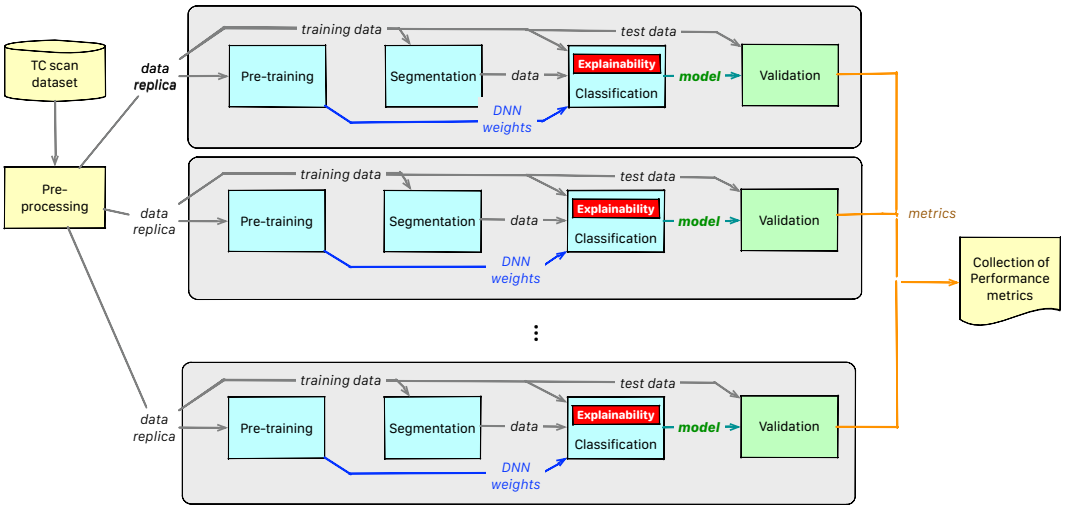

Figure 5 CLAIRE-COVID19 universal pipeline unfolded.

are at the frontier of EU digital sovereignty, which, together with the green new deal, are two cornerstones of the EU agenda.

With EU 8B€ funding, HPC ranks first for the funding volume in the EU Digital Europe 2021-27. Propelled by Artificial Intelligence, the HPC market analysis reports an overall CAGR19-24 of 32.9%[5], where health is among the driver domains, and where the high-end HPC (supercomputing) market grows much faster than other segments. However, Europe struggles to mature an HPC value chain, from advanced research to innovation.

The HPC ecosystem is partitioned into applications, system software, and infrastructures. We believe that the mainstream industrial adoption of HPC requires a system software part, enabling technology to transform applications into easily usable services hence into innovation. While in scientific computing the modernisation of HPC applications is a scientific desideratum required to boost industrial adoption, in AI the shift toward the Cloud model of services is a must. AI applications are already modern, and they will not step back.

In the HPC landscape, AI requires a significant shift in current software technology. Computing resources should be available on-demand as a service, and data should be kept secure in public and shared infrastructures. StreamFlow, leveraging modern virtualisation technologies, advocates a new methodology to assemble existing legacy codes in a portable and malleable way.

In this work, we propose the cluster-as-accelerator design pattern as a way to enable HPC applications cloudification, allowing practitioners to minimise the price-performance ratio. Moreover, we advocate hybrid workflow models as an intuitive programming paradigm to express such design pattern, reducing technical barriers to HPC facilities for domain experts without a strong computer science background.

The CLAIRE-COVID19 universal pipeline has been designed according to these principles, offloading training-related steps to an HPC centre and collecting back the resulting networks on the host execution flow, located on a Cloud infrastructure, for visualisation purposes.

[5] Compound Annual Growth Rate.
Source: `https://orau.gov/ai_townhall/presentations/1115am-Hyperion_Research_AI_Research.pdf`

The StreamFlow toolkit, a Workflow Management System supporting hybrid workflow executions, has been successfully used to perform a first portion of the planned experiments, proving the proposed approach's effectiveness in the AI domain.

The experiments will continue to execute all the variants on HPC4AI, applying the pipeline on the complete dataset. However, the main goal will be setting a comprehensive framework, where StreamFlow could manage the pipeline execution on different hybrid HPC infrastructures.

References

1 Enis Afgan, Dannon Baker, Bérénice Batut, Marius van den Beek, Dave Bouvier, Martin Cech, John Chilton, Dave Clements, Nate Coraor, Björn A. Grüning, Aysam Guerler, Jennifer Hillman-Jackson, Saskia D. Hiltemann, Vahid Jalili, Helena Rasche, Nicola Soranzo, Jeremy Goecks, James Taylor, Anton Nekrutenko, and Daniel J. Blankenberg. The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2018 update. *Nucleic Acids Res.*, 46(Webserver-Issue):W537–W544, 2018. doi:10.1093/nar/gky379.

2 Michael Albrecht, Patrick Donnelly, Peter Bui, and Douglas Thain. Makeflow: a portable abstraction for data intensive computing on clusters, clouds, and grids. In *Proceedings of the 1st ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies, SWEET@SIGMOD 2012, Scottsdale, AZ, USA, May 20, 2012*, page 1, 2012. doi:10.1145/2443416.2443417.

3 Marco Aldinucci. High-performance computing and AI team up for COVID-19 diagnostic imaging. https://aihub.org/2021/01/12/high-performance-computing-and-ai-team-up-for-covid-19-diagnostic-imaging/, January 2021. Accessed: 2021-01-25.

4 Marco Aldinucci, Marco Danelutto, Lorenzo Anardu, Massimo Torquati, and Peter Kilpatrick. Parallel patterns + macro data flow for multi-core programming. In *Proc. of Intl. Euromicro PDP 2012: Parallel Distributed and network-based Processing*, pages 27–36, Garching, Germany, February 2012. IEEE. doi:10.1109/PDP.2012.44.

5 Marco Aldinucci, Sergio Rabellino, Marco Pironti, Filippo Spiga, Paolo Viviani, Maurizio Drocco, Marco Guerzoni, Guido Boella, Marco Mellia, Paolo Margara, Idillio Drago, Roberto Marturano, Guido Marchetto, Elio Piccolo, Stefano Bagnasco, Stefano Lusso, Sara Vallero, Giuseppe Attardi, Alex Barchiesi, Alberto Colla, and Fulvio Galeazzi. HPC4AI, an AI-on-demand federated platform endeavour. In *ACM Computing Frontiers*, Ischia, Italy, May 2018. doi:10.1145/3203217.3205340.

6 Peter Amstutz, Michael R. Crusoe, Nebojša Tijanić, Brad Chapman, John Chilton, Michael Heuer, Andrey Kartashov, John Kern, Dan Leehr, Hervé Ménager, Maya Nedeljkovich, Matt Scales, Stian Soiland-Reyes, and Luka Stojanovic. Common workflow language, v1.0, 2016. doi:10.6084/m9.figshare.3115156.v2.

7 Michael R. Berthold, Nicolas Cebron, Fabian Dill, Thomas R. Gabriel, Tobias Kötter, Thorsten Meinl, Peter Ohl, Christoph Sieb, Kilian Thiel, and Bernd Wiswedel. KNIME: the Konstanz Information Miner. In *Data Analysis, Machine Learning and Applications - Proceedings of the 31st Annual Conference of the Gesellschaft für Klassifikation e.V., Albert-Ludwigs-Universität Freiburg, March 7-9, 2007*, Studies in Classification, Data Analysis, and Knowledge Organization, pages 319–326. Springer, 2007. doi:10.1007/978-3-540-78246-9_38.

8 Iacopo Colonnelli, Barbara Cantalupo, Ivan Merelli, and Marco Aldinucci. StreamFlow: cross-breeding cloud with HPC. *IEEE Transactions on Emerging Topics in Computing*, August 2020. doi:10.1109/TETC.2020.3019202.

9 Rafael Ferreira da Silva, Rosa Filgueira, Ilia Pietri, Ming Jiang, Rizos Sakellariou, and Ewa Deelman. A characterization of workflow management systems for extreme-scale applications. *Future Gener. Comput. Syst.*, 75:228–238, 2017. doi:10.1016/j.future.2017.02.026.

10 Ewa Deelman, Karan Vahi, Gideon Juve, Mats Rynge, Scott Callaghan, Philip Maechling, Rajiv Mayani, Weiwei Chen, Rafael Ferreira da Silva, Miron Livny, and R. Kent Wenger. Pegasus, a workflow management system for science automation. *Future Generation Comp. Syst.*, 46:17–35, 2015. doi:10.1016/j.future.2014.10.008.

11 Paolo Di Tommaso, Maria Chatzou, Evan W. Floden, Pablo P. Barja, Emilio Palumbo, and Cedric Notredame. Nextflow enables reproducible computational workflows. *Nature Biotechnology*, 35(4):316–319, April 2017. doi:10.1038/nbt.3820.

12 Maurizio Drocco, Claudia Misale, and Marco Aldinucci. A cluster-as-accelerator approach for SPMD-free data parallelism. In *Proc. of 24th Euromicro Intl. Conference on Parallel Distributed and network-based Processing (PDP)*, pages 350–353, Crete, Greece, 2016. IEEE. doi:10.1109/PDP.2016.97.

13 Thomas Fahringer, Radu Prodan, Rubing Duan, Jürgen Hofer, Farrukh Nadeem, Francesco Nerieri, Stefan Podlipnig, Jun Qin, Mumtaz Siddiqui, Hong Linh Truong, Alex Villazón, and Marek Wieczorek. ASKALON: A development and grid computing environment for scientific workflows. In *Workflows for e-Science, Scientific Workflows for Grids*, pages 450–471. Springer, 2007. doi:10.1007/978-1-84628-757-2_27.

14 Johannes Köster and Sven Rahmann. Snakemake - a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 2012. doi:10.1093/bioinformatics/bts480.

15 Michael Kotliar, Andrey V Kartashov, and Artem Barski. CWL-Airflow: a lightweight pipeline manager supporting Common Workflow Language. *GigaScience*, 8(7), July 2019. doi:10.1093/gigascience/giz084.

16 E.A. Lee and T.M. Parks. Dataflow process networks. *Proc. of the IEEE*, 83(5):773–801, May 1995. doi:10.1109/5.381846.

17 Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew B. Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience*, 18(10):1039–1065, 2006. doi:10.1002/cpe.994.

18 Claudia Misale, Maurizio Drocco, Marco Aldinucci, and Guy Tremblay. A comparison of big data frameworks on a layered dataflow model. *Parallel Processing Letters*, 27(01):1–20, 2017. doi:10.1142/S0129626417400035.

19 Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I. Jordan, and Ion Stoica. Ray: A distributed framework for emerging AI applications. In *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pages 561–577, 2018.

20 Enric Tejedor, Yolanda Becerra, Guillem Alomar, Anna Queralt, Rosa M. Badia, Jordi Torres, Toni Cortes, and Jesús Labarta. PyCOMPSs: Parallel computational workflows in Python. *Int. J. High Perform. Comput. Appl.*, 31(1):66–82, 2017. doi:10.1177/1094342015594678.

21 John Vivian, Arjun A. Rao, Frank A. Nothaft, et al. Toil enables reproducible, open source, big biomedical data analyses. *Nature Biotechnology*, 35(4):314–316, April 2017. doi:10.1038/nbt.3772.

22 Justin M. Wozniak, Michael Wilde, and Ian T. Foster. Language features for scalable distributed-memory dataflow computing. In *Proceedings of the 2014 Fourth Workshop on Data-Flow Execution Models for Extreme Scale Computing*, DFM '14, page 50–53, USA, 2014. IEEE Computer Society. doi:10.1109/DFM.2014.17.

23 Matei Zaharia, Reynold S. Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J. Franklin, Ali Ghodsi, Joseph Gonzalez, Scott Shenker, and Ion Stoica. Apache Spark: a unified engine for big data processing. *Commun. ACM*, 59(11):56–65, 2016. doi:10.1145/2934664.