Scientific workflows on clouds with heterogeneous and preemptible instances

Scientific Workflows on Clouds with Heterogeneous and Preemptible Instances

Fabio TORDINI [a,1], Marco ALDINUCCI [a], Paolo VIVIANI [a], Ivan MERELLI [b], Pietro LIÒ [c]

[a] *Computer Science Dept, University of Torino, Torino, Italy*
[b] *IBT – CNR, Segrate (MI), Italy*
[c] *Computer Laboratory, University of Cambridge, Cambridge, UK*

Abstract. The cloud environment is increasingly appealing for the HPC community, which has always dealt with scientific applications. However, there is still some skepticism about moving from traditional physical infrastructures to virtual HPC clusters. This mistrusting probably originates from some well known factors, including the effective economy of using cloud services, data and software availability, and the longstanding matter of data stewardship. In this work we discuss the design of a framework (based on Mesos) aimed at achieving a cost-effective and efficient usage of *heterogeneous* Processing Elements (PEs) for workflow execution, which supports *hybrid cloud bursting* over preemptible cloud Virtual Machines.

Keywords. HPC, Scientific Workflows, Cloud Engineering, Resource Provisioning

1. Introduction

In the HPC landscape, workflows play a very important role for scientific computing and application coordination, because they provide means to formalize and organize complex scientific processes by supporting the modeling, execution and monitoring of the data analysis process. In this paper we mainly focus on Bioinformatics workflows, commonly referred as *pipelines*. They typically exploit a pure *Data flow* behavior [7].

For scientists to switch to the cloud for their analysis and computations, an important metric to consider is the cost for resource usage, taking into account application's workload and performance requirements. Typically, the cost per unit of time of Infrastructure-as-a-Service (IaaS) grows in a linear-affine fashion with Virtual Machines (VMs) reliability, core count, memory size and storage (in increasing order of weight); VMs executing tasks using only a fraction of their cores are entirely payed. Bioinformatics pipelines are typically long-running and performance-demanding applications. Nowadays, their execution on cloud might be a quite expensive option.

Notwithstanding, the cloud makes it possible to execute a scientific workflow with a close to zero investment in infrastructures, according to the pay-per-use model. Also, the cloud execution model offers a much more dynamic execution model with respect to traditional HPC clusters. With their technically unbound resource availability, several pub-

[1] Corresponding Author: tordini@di.unito.it

lic cloud providers offers the *elasticity* feature, meaning that VM count can be scaled up or down to adapt to workload variations over independent tasks. The idea of elasticity is normally bound to *auto-scaling* techniques, consisting in pre-configured scaling policies. While this concept mostly applies to uniform clusters, we envision elasticity coupled with a complementary adaptive mechanism specifically designed to reduce the cost of execution of scientific workflows in cloud environments. We call this feature *plasticity*. It is designed to support the execution of workflows on a *hybrid cloud*, i.e. heterogeneous clusters of VMs with different size and reliability. We will show that this makes possible to execute the workflow using generally smaller and cheaper (i.e. preemptive) VMs, while guaranteeing the correct termination of the workflow execution. Specifically, we distinguish PEs using three *plasticity features*: 1) core count – $\#vcpus \in \mathbb{N}$, 2) memory size – $\#vram \in \mathbb{N}$, and 3) preemptibility – $preempt \in \{T, F\}$, which is the possibility of sudden and unavoidable termination.

In this work we focus on plasticity as mechanism for adaptive execution of workflows that complement elasticity. We give a definition of an execution model of a hybrid cloud, and we explain how plasticity helps in defining schedule and provisioning strategies that support the execution of scientific workflows over heterogeneous hybrid clusters. We also depict the components that support our execution model, describing their role in maintaining ad adaptive infrastructure that can efficiently host tasks execution, adhering to user's QoS requirements. Section 2 provides a background on scientific workflows with a focus on the existing facilities for their description and execution; in Section 3 the hybrid cloud model is defined, explaining the concepts that support plasticity; Section 4 illustrates the main components of our infrastructure and describes their implementation; Section 5 concludes this work and gives some details about future perspectives.

2. Background

Different stages of a scientific workflow are just fundamentally different, and have different parallelism, memory access and data access requirements. The cloud paradigm can help in addressing these requirements [5]; for instance, *on-demand* resources reflect a pay-per-usage model. Many cloud vendors provide also *preemptible* resources, that allow to run cloud instances at a much lower price, with the drawback that such instances can be terminated by the service provider.

In this section we will review some core aspects upon which this work deploys, starting from the description of workflows to their execution on distributed computing environments, with a discussion of related works.

2.1. Workflow description

A workflow is modeled as a process made up of multiple steps or *tasks* arranged as a Directed Acyclic Graph (DAG), where each node of the DAG is an activity that needs to be conducted and that is characterized by a number of *parameters*, which are named input or output place-holders with an associated data-type or schema. Also, each task can be characterized with extra-functional attributes, such as plasticity features. Tasks output are connected to the inputs of downstream steps. They are DAG edges and represent data-

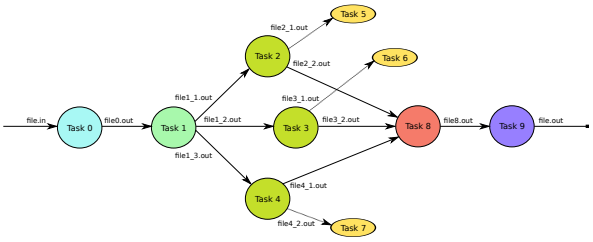

Figure 1. Sample workflow with ten tasks. Nodes represent computational tasks, edges are data dependencies between tasks

and control-flow dependencies between tasks (see Figure 1). A task becomes executable (or *fireable*) as soon as all input parameters are available. At any point in time, all fireable tasks are independent and can be executed in parallel.

One effort to standardize workflow description and analysis tools – in a way that makes them portable across a variety of software and hardware environments – is the *Common Workflow Language* (CWL) [6]: workflow steps are described using CWL's Domain Specific Language (DSL), which details inputs, outputs and commands to execute. Such description can include annotations that qualify the job type and job's resource requirements, helping job schedulers to best accommodate these job types. Similar approaches to a standard formal description language for workflows include the *Workflow Definition Language* (WDL) [23] and *Yet Another Workflow Language* (YAWL) [21].

2.2. Workflow execution and task scheduling

Workflow execution on distributed HPC platforms requires a *scheduler* able to dispatch tasks to computing resources. Here we introduce some basic concepts of job scheduling and discuss some of the existing schedulers. A review of HPC job schedulers, with a classification and a comparison of the most representative ones, can be found in [19].

A scheduler (also known as batch system, Resource Manager System, or workload automation) is responsible for managing the tasks listed in a job queue, in which jobs wait to be scheduled. Job queues may include different types of jobs, each characterized by different priorities, estimated execution time and resource requirements.

A job scheduler is thus in charge of: 1) manage job placement into computing resources; 2) run the job; 3) report the outcome of the execution; 4) provide a failure recovery mechanism.

Schedulers are a primary component of a scalable computing infrastructure, because they control all the work on the system and directly impact the overall system efficiency. Traditional HPC schedulers include *PBS* [12], *GridEngine* [11], *HTCondor* [20] and *Slurm* [25]. These schedulers are fully featured tools that provide queues and resources management and job scheduling features. Each of them has peculiar features that make it better suitable for specific workloads.

On the other end, *Apache Mesos* [13] makes resources available for scheduling and carries out the logistics of launching and monitoring tasks: in a similar way that an operating system manages access to the resources on a desktop computer, Mesos enables the partitioning of a pool of compute resources among many scheduling domains, and ensures that applications have access to the resources they need.

2.3. Related works

There exist several frameworks and libraries for workflow description and execution: a reasoned list of existing tools, with a discussion and classification of workflow management solutions is given in [17].

Many tools implement abstract workflow description languages through general-purpose language bindings (such as Python, Java, etc.), enhancing expressiveness and portability and making the abstract representation an executable application. Among such frameworks, we mention *Nextflow* [9], *Snakemake* [16], and *Toil* [22]. Using these packages, workflows can be written upon general-purpose programming languages, though exposing a different specific grammar for workflow definition, and possibly allowing in-line code of various scripting languages.

The majority of frameworks are built upon implicit wild-card rules definition (like in a Makefile), but use high-level languages to improve logic functionality and code readability. They allow to define file transformation rules, while the framework engine builds the entire topology upon execution. Differently, Toil is a class-based framework, which inherits the OOP paradigm. It is a Python workflow engine that offers explicit APIs for defining task dependencies from within task methods. It is also one of the few frameworks providing extensive support for CWL, including facilities for workflow analysis and execution on batch systems.

Workflow Management Systems (WMS) bundle together the features listed above, providing a high-level (often graphical) tool that is agnostic of the physical resources where the workflow is executed. Among them are *Galaxy* [1], *Pegasus* [8] and *Taverna* [18]. In particular, Galaxy is a server/cloud workbench that provides a graphical design panel and extensive computing and storage facilities, all accessible through a friendly, web-based user interface.

Some studies have investigated cost minimization on public cloud platforms. Some of them focus on Amazon EC2 spot (i.e. preemptible) instances: an important work proposes statistical models to help users bid resources by fitting spot prices and time between price changes [14]; another interesting work investigates check-pointing mechanisms to minimize costs while maximizing the reliability with spot instances [24].

Among the tools and frameworks discussed and referenced, some of them provide facilities for the execution of workflows on cloud resources. While most cloud vendors provide auto-scaling techniques, they mostly apply to homogeneous cloud clusters (same hardware features and pricing model).

Due to their dynamic nature and the heterogeneity of the tasks they encompass, we believe that scientific workflows better run on heterogeneous clusters, where different types of hardware configurations are desirable to meet jobs' computing requirements, while taking advantage of the cloud's pricing models and helping reducing infrastructure costs. To the best of our knowledge, none of the existing frameworks takes into account this heterogeneity requirement, and we will illustrate how it can be achieved and the benefits it can bring to the execution of a workflow.

3. Plastic workflows on the hybrid cloud

3.1. Execution model: The hybrid cloud

The core of the present work consists in the design and implementation of a scheduler supporting the execution of workflows across a hybrid cloud, which is defined as an *heterogeneous* cluster of *homogeneous* clusters of processing elements (PEs). In turn, each PE is described by the previously mentioned features #*vcpus*, #*vram*, *preempt*. Also, PEs in a cluster can be either VMs or on-premises resources. We can distinguish:

- Hybrid Cloud $HC = \{CC_1, CC_2, \dots CC_n\}$
- Homogeneous Cluster $CC_i = \{PE^i_1, PE^i_2, \dots PE^i_k\}$, where each PE^i_j exhibits the same #*vcpu*, #*vram*, and *preempt*; they are all either VMs or not.

The composition of *HC* is defined at launch time and remains constant during the execution of the workflow. According to HPC lexicon, it is a *moldable* (i.e. easy o be modeled) job. Notwithstanding, each PE^i_j marked with $preempt = T$ can be terminated in any moment of the execution. This scenario differs from elastic cloud scenarios for three key issues: 1) resources are heterogeneous, 2) resource count is non-increasing, 3) execution strategy reacts to unexpected infrastructure changes (e.g. preemption, unbalanced workload).

We consider workflows described in CWL, where plasticity features can be annotated for each task: 1) core count – $\#vcpus \in \mathbb{N}$, 2) memory size – $\#vram \in \mathbb{N}$, and 3) preemptibility – $preempt \in \{T, F\}$. The scheduler matches these target features against all CC_i to make a greedy scheduling decision.

Workflow tasks have no deadline. For this reason the adopted schedule strategy is on-line and best-effort. It aims at scheduling any given task $t_i \in T$ onto the next likely unused PE^i_j that best fits the requirements of t_i. This mapping is easily obtainable through a (simple) function of $\#vcpu, \#vram$, and *preempt*. The very same approach can be used to define more sophisticated global optimization strategies, which will necessarily exhibit a stochastic nature because of the non-deterministic behavior of preemption.

The total cost for workflow execution on a cloud cluster is linked to the execution time of each task t_i in each PE^i_j: cloud resources follow a pay-as-you-go model, where VMs are charged per billing periods. Follows that the longer a task time span, the higher the cost for resource leasing. When execution time is a crucial requirement, costs inevitably increase because we need to use dedicated, high-end resources, which offer higher reliability at a higher price. On the other hand, when costs must be kept at the minimum, we can give up performance but choose less powerful (and less reliable) configurations.

For the sake of simplicity, in this work we do not deal with data movement and storage issues: although these factors play a fundamental role in system performance and actual infrastructure costs, we temporary omit these elements and assume that all data is stored in a global storage facility (e.g., Amazon *S3*, *Dropbox*). Computing nodes should be able to get snapshots of the data and read it, while the results are put back to the same storage, in an structured manner.

3.2. Programming model: annotated CWL

Scientific workflows exhibit recurrent features that have an effective impact on the execution of the involved tasks, such as data distribution and data aggregation. Their full characterization is presented in [15]. They also present varied data and computational characteristics, including:

1. Tasks filter or transform data, and can be either quick or long-lived. Also, tasks are either sequential or multi-threaded.
2. The whole workflow, or a part of it, can be executed on a batch of independent data.

These two characteristics can be used to label workflow tasks for a plastic execution. The former one can be exploited to size the task, whereas the second is useful for marking preemption feature.

Setting #*vcpus* and #*vram* features can be hardly made automatic, since the reasonable values for these features depend on the knowledge of the domain and the tools used in each specific task. Here the workflow designers are asked to size the task, i.e. imagine the minimum compute capability (#*vcpus* and #*vram*) each specific task requires. The workflow approach simplifies the process since it decouples the evaluation of compute capability task by task. A very common case are inherently sequential tasks and simple data filtering that typically require a "small" VM, e.g. $\#vcpus = 1$. If the task supports multi-threading, #*vcpus* can be also used to set the task's parallelism degree, which can be often controlled with launch time parameters.

A more automatic reasoning can be made on preemption: it is useful to explicitly declare in the workflow all the cases in which functional replication is used, i.e. where a part of the workflow is repeatedly applied to a bulk of independent data, meaning that they are farmed out [4,3]. We assume to use two special workflow nodes (*Co-Start/Co-End*) to delimit the part of the workflow subject to replication, collectively called *worker* macro-node. Co-Start/Co-End nodes trammel workers in a lattice fashion. An example is shown in Fig. 2. The execution of the whole workflow on a bulk of independent data is a common case. In this case the whole workflow will result entirely contained between two special nodes in a lattice.

Co-Start/Co-End nodes permit to compactly represent a parametric number of replicas of the worker node instances (without really adding nodes to the description of the DAG) [2]. Worker nodes are good candidates to be marked with $preempt = T$ features, since they are independent, i.e. the failure of one of them does not directly affect any other node, apart from the collector, which can be executed only when all the instances of the worker nodes has been successfully executed. For this reason, the preemption of VMs executing worker node instances is a non-catastrophic failure. To accomplish this, the scheduler should periodically monitor the execution of worker node instances assigned to preemptible VMs, and reissue failed tasks onto different VMs, that can reliably run the task to successful termination. It can be useful to define $preempt \in \mathbb{N}, 0 < preempt \leq \infty$, representing the marker cost at which the VM instance will be preempted, and where ∞ denotes on-demand VMs.

Observe that the definition of a balanced hybrid cloud significantly influences the effectiveness of the approach, which actually depends on a well-designed mix of clusters with complementary features. As an example:

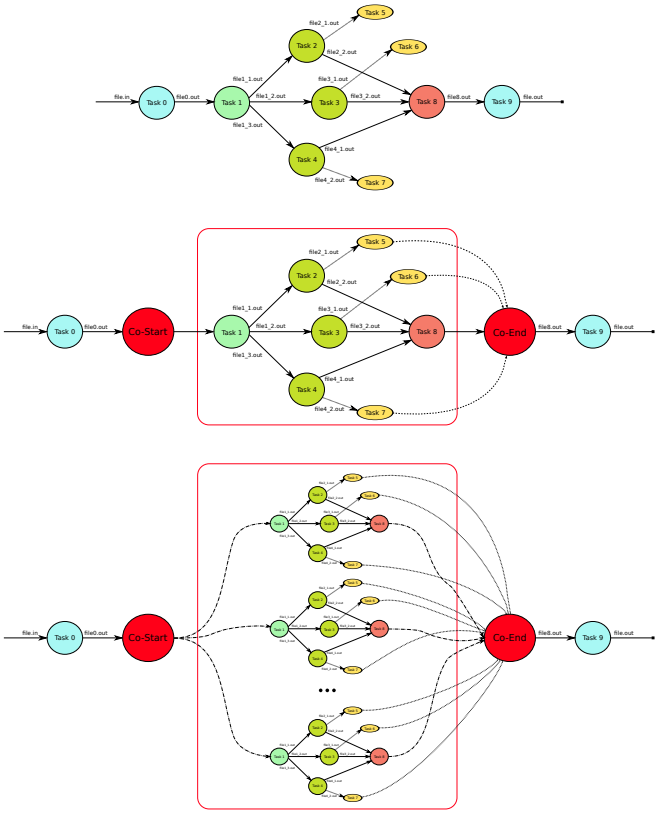

Figure 2. Functional replication of workflow steps: *top* graph shows a generic workflow; *middle* drawing identifies a subset of steps good for replication; *bottom* drawing shows a functional replication of a number of steps over a bulk of data. Red nodes drive steps firing, acting as synchronization points

Core A cluster of on-demand ($preempt = \infty$) large VMs, e.g. $\#vcpus = 8 - 16$, and enough $\#vram$ for the most memory demanding task. According to the Hybrid Cloud paradigm, reliable VMs can be substituted with on-premise PEs.

Burst-large A cluster of preemptible VMs ($preempt = x$) large VMs, e.g. $\#vcpus = 8 - 16$.

Burst-small A cluster of preemptible VMs ($preempt = x$) small VMs, e.g. $\#vcpus = 1$

The two burst clusters can be substituted with more PEs exhibiting different *preempt* levels, thus a different likelihood of failure during execution. Eventually, the performance-cost trade-off can be fixed by the Hybrid Cloud composition, while the correctness guarantee is enforced by the presence of reliable (at least one) VMs and the scheduling strategy.

4. Achieve plasticity on heterogeneous clusters

To implement our solution we use Toil for its extensive support for CWL, and Mesos as a resource orchestrator. The main components of the system are the *resource manager*,

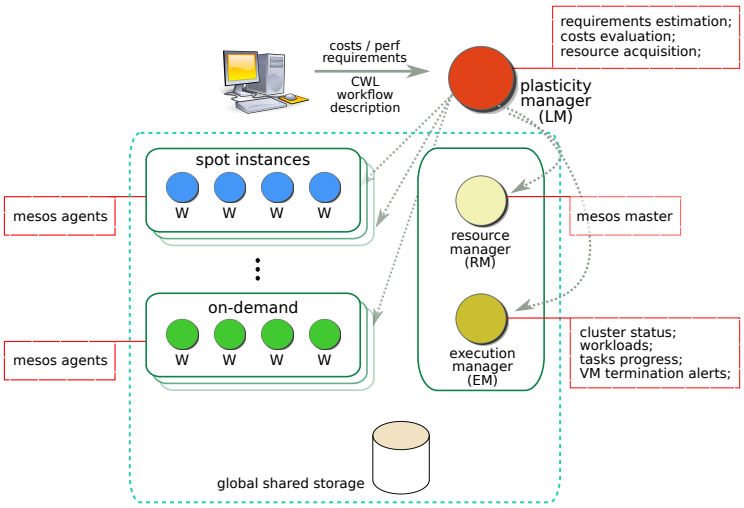

Figure 3. An overview of our solution

the *plasticity manager* and the *execution manager* (see Figure 3). As already described in section 3, we assume the presence of a global storage facility shared and accessible by all components, where the state of the workflow is maintained. Each job is backed by a file in the storage, and atomic updates are used to ensure the workflow can always be resumed upon failure. All user files and datasets are also maintained there, allowing them to be shared between jobs.

The resource manager corresponds to a "master" in Mesos terminology, and has knowledge about all available PEs (called *agents* in Mesos): it consists of a Mesos framework that *offer* resources to applications willing to execute and schedules tasks among available computing components.

The plasticity manager parses the workflow description and performs an initial analysis of the computing requirements for each of the involved steps, estimating the amount of computing resources required for the workflow to be executed, taking into account the budget specified. The decision is based on the evaluation of multiple factors, including the computing requirements annotated in each task's definition, the available computing resources, the estimated costs for the whole computing infrastructure and users' QoS specifications.

Based on the metrics collected from the initial analysis, scheduling and provisioning strategies are built, which proceed by deploying the required computing infrastructure that better suits for the requirements computed. The provisioning scheme results from a greedy approach and does not seek to obtain an optimal solution to the provisioning problem (which is known to be NP-hard [10]).

The execution manager supervises running agents, checking for under- or over-provisioning issues and task failures. It maintains information about jobs status and reacts to failures by re-scheduling the failed job. At every time span it polls preemptible PEs to make sure they have not been signalled to shut down by the cloud provider. If that should happen – a warning message is sent by the cloud provider when it is about to happen – jobs running there need to be rescheduled to another instance.

During workflow analysis and evaluation, tasks are ordered according to a priority score. The dominant sorting criteria is preemptibility: jobs marked as $preempt = F$ (e.g., critical tasks) should be considered first for execution (keeping data dependencies) because they can only be run on standard nodes, while other jobs can run on both. Using a first-fit decreasing, bin-packing-like algorithm we can calculate an approximate minimum number of PEs that will fit a given list of jobs.

5. Conclusions and perspectives

In this work we have outlined our ideas concerning a plasticity mechanism, intended to support the execution of scientific workflows in hybrid cloud environments. Plasticity provides facilities for an adaptive execution of workflows in heterogeneous environments, characterized by an heterogeneous cluster of homogeneous clusters of processing elements, where each processing element might belong to a different level of reliability: it is the case of preemptible instances, that can be acquired at much lower prices, with the risk of having the shut down at provider's needs.

By coupling the underlying logic of plasticity with proper task scheduling and dynamic resource provisioning, it is possibly to better exploit the cloud paradigm for workflow execution, while keeping the expenses on budget. We showed that the presence of at lest one reliable instance guarantees the correctness of the execution, while the performance-costs trade-off will affect the hybrid cloud composition.

Studies on scientific workflows and HPC exploiting cloud resources are central to a large range of data profiling areas. This is increasingly true as data from different fields will need similar processing and databases keep growing. The combination of cloud computing and optimisation of workflows has two important benefits: improved resource utilization, which translates into cautious and affordable costs for running complex scientific applications; improved performance, derived from the reasoned mapping of tasks on best fitting resources.

Considering Bioinformatics, scientific pipelines (i.e. workflows) are widely used for data analysis purposes over huge sets of heterogeneous raw data: in order for results to be significant, it is not uncommon to run the same stage of an analysis in a number of different ways, to demonstrate the robustness of novel results, or to tackle different sorts of data, that have undergone different perturbations or that have been collected at different time points. And the final result may be the aggregation of the outcomes of these independent workflows.

Acknowledgements

This work was partially supported by the EU H2020 project RePhrase (no. 644235) and by the OptiBike experiment of the EU I4MS Fortissimo2 (no. 680481).

References

[1] E. Afgan, D. Baker, M. van den Beek, et al. The galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update. *Nucleic Acids Research*, 44(W1):W3–W10, 2016.

[2] M. Aldinucci, S. Campa, M. Danelutto, P. Kilpatrick, and M. Torquati. Targeting distributed systems in fastflow. In *Proceedings of the 18th International Conference on Parallel Processing Workshops*, Euro-Par'12, pages 47–56, Berlin, Heidelberg, 2013. Springer-Verlag.

[3] M. Aldinucci, S. Campa, M. Danelutto, P. Kilpatrick, and M. Torquati. Design patterns percolating to parallel programming framework implementation. *International Journal of Parallel Programming*, 42(6):1012–1031, Dec 2014.

[4] M. Aldinucci, M. Danelutto, M. Meneghin, M. Torquati, and P. Kilpatrick. *Efficient streaming applications on multi-core with FastFlow: The biosequence alignment test-bed*, volume 19 of *Advances in Parallel Computing*. Elsevier, 2010.

[5] M. Aldinucci, M. Torquati, C. Spampinato, M. Drocco, C. Misale, C. Calcagno, and M. Coppo. Parallel stochastic systems biology in the cloud. *Briefings in Bioinformatics*, 15(5):798–813, 2014.

[6] P. Amstutz, M. R. Crusoe, N. Tijani, et al. *Common Workflow Language, v1.0*, 2016.

[7] V. Curcin and M. Ghanem. Scientific workflow systems - can one size fit all? In *2008 Cairo International Biomedical Engineering Conference*, pages 1–9, Dec. 2008.

[8] E. Deelman, K. Vahi, G. Juve, et al. Pegasus: a workflow management system for science automation. *Future Generation Computer Systems*, 46:17–35, 2015.

[9] P. Di Tommaso, M. Chatzou, E. W. Floden, et al. Nextflow enables reproducible computational workflows. *Nature Biotechnology*, 35(4):316–319, Apr. 2017.

[10] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[11] W. Gentzsch. Sun Grid Engine: Towards Creating a Compute Power Grid. In *Proc. of the 1st Intl. Symposium on Cluster Computing and the Grid*, CCGRID, pages 35–, Washington, DC, USA, 2001. IEEE Computer Society.

[12] R. L. Henderson. *Job scheduling under the Portable Batch System*, pages 279–294. Springer Berlin Heidelberg, Berlin, Heidelberg, 1995.

[13] B. Hindman, A. Konwinski, M. Zaharia, et al. Mesos: A platform for fine-grained resource sharing in the data center. In *Proc. of the 8th USENIX Conference on Networked Systems Design and Implementation*, NSDI'11, pages 295–308, Berkeley, CA, USA, 2011. USENIX Association.

[14] B. Javadi, R. K. Thulasiramy, and R. Buyya. Statistical modeling of spot instance prices in public cloud environments. In *Proc of the 4th IEEE Intl. Conference on Utility and Cloud Computing*, UCC, pages 219–228, Washington, DC, USA, 2011. IEEE.

[15] G. Juve, A. Chervenak, E. Deelman, et al. Characterizing and Profiling Scientific Workflows. *Future Generation Computer Systems*, 29(3):682–692, Mar. 2013.

[16] J. Kster and S. Rahmann. Snakemake - a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 2012.

[17] J. Leipzig. A review of bioinformatic pipeline frameworks. *Briefings in Bioinformatics*, 18(3):530–536, 2017.

[18] T. Oinn, M. Greenwood, M. Addis, et al. Taverna: Lessons in creating a workflow environment for the life sciences: Research articles. *Concurrency and Computation: Practice and Experience*, 18(10):1067–1100, Aug. 2006.

[19] A. Reuther, C. Byun, W. Arcand, and other. Scalable system scheduling for HPC and big data. *CoRR*, abs/1705.03102, 2017.

[20] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: the condor experience. *Concurrency and Computation: Practice and Experience*, 17(2-4):323–356, 2005.

[21] W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: Yet Another Workflow Language. *Inf. Syst.*, 30(4):245–275, June 2005.

[22] J. Vivian, A. A. Rao, F. A. Nothaft, et al. Toil enables reproducible, open source, big biomedical data analyses. *Nature Biotechnology*, 35(4):314–316, Apr. 2017.

[23] *Workflow Description Lnaguage (WDL)*. Accessed: 2017-08-01.

[24] S. Yi, D. Kondo, and A. Andrzejak. Reducing costs of spot instances via checkpointing in the Amazon Elastic Compute Cloud. In *Intl. Conference on Cloud Computing*, pages 236–243. IEEE, July 2010.

[25] A. B. Yoo, M. A. Jette, and M. Grondona. *SLURM: Simple Linux Utility for Resource Management*, pages 44–60. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.