The BioExcel methodology for developing dynamic, scalable, reliable and portable computational biomolecular workflows

Jorge Ejarque*, Pau Andrio*, Adam Hospital†, Javier Conejero*,
Daniele Lezzi*, Josep LL. Gelpi*‡, Rosa M. Badia*
*Barcelona Supercomputing Center (BSC-CNS), Barcelona, Spain
†Institute for Research in Biomedicine (IRB), Barcelona, Spain
‡Dept. Biochemistry and Molecular Biomedicine, University of Barcelona, Spain
jorge.ejarque@bsc.es, pau.andrio@bsc.es, adam.hospital@irbbarcelona.org, francisco.conejero@bsc.es,
daniele.lezzi@bsc.es, josep.gelpi@bsc.es, rosa.m.badia@bsc.es

Abstract—**Developing complex biomolecular workflows is not always straightforward. It requires tedious developments to enable the interoperability between the different biomolecular simulation and analysis tools. Moreover, the need to execute the pipelines on distributed systems increases the complexity of these developments. To address these issues, we propose a methodology to simplify the implementation of these workflows on HPC infrastructures. It combines a library, the BioExcel Building Blocks (BioBBs), that allows scientists to implement biomolecular pipelines as Python scripts, and the PyCOMPSs programming framework which allows to easily convert Python scripts into task-based parallel workflows executed in distributed computing systems such as HPC clusters, clouds, containerized platforms, etc. Using this methodology, we have implemented a set of computational molecular workflows and we have performed several experiments to validate its portability, scalability, reliability and malleability.**

I. Introduction

Computational workflows are one of the most used tools to assemble and run simulations of different scientific fields as climate predictions, bioinformatics, engineering, etc. Researchers can compose their applications, usually made of pieces of code available in libraries and binaries, using a textual or graphical representation of the dependencies between those parts, and let the runtime of the workflow management system to orchestrate the execution on a given computational platform. In particular, HPC systems are becoming more attractive for the execution of workflows that traditionaly have been executed on distributed systems as grids or clouds, because they can have tasks that require a certain degree of massive parallelism (i.e., OpenMP/MPI tasks, GPUs). The trend is to have complex HPC systems built on hybrid architectures that combine traditional processors with accelerators and other devices. On top of the computing complexity, the packaging of workflows is an additional issue, with containers becoming a popular way to distribute and deploy applications.

To address the issues above, it is a must to have a system that can offer a simple interface for the composition of applications components which are able to transparently manage their execution, adapting them to the different capabilities of heterogeneous computing and to the dynamicity of the computational load.Moreover, the computation requirements of these workflows impose to scale their execution to a large amount of resources and to provide reliability mechanisms.

This paper presents a methodology for defining and orchestrating biomolecular simulations on HPC infrastructures that satisfies the above mentioned requirements. It combines a software library developed by the BioExcel Centre of Excellence, BioExcel Building Blocks (BioBBs) [1], with a workflow programming framework, PyCOMPSs [2] [3]. On the one hand, the BioBB library allows scientist to implement pipelines as Python scripts that automatize the various steps of Molecular Dynamics (MD) simulations that are performed manually in many cases. On the other hand, PyCOMPSs converts Python scripts into task-based parallel workflows and orchestrates the execution of the computational tasks in resources of distributed computing systems such as HPC clusters, clouds or containerized platforms [4]. Using this methodology, we have implemented a set of computational molecular workflows and performed several experiments to validate its portability, scalability, reliability and elasticity.

The paper is structured as follows: Section III describes the proposed methodology; Section IV presents the workflows implemented adopting this methodology; Section V reports the validation of the features provided by the presented methodology; Section II presents the state of the art and related work on topics involved in the proposed research. Finally, Section VI draws the conclusion and proposes guidelines for future research in this topic.

II. Related Work

The use of computational workflows has become ubiquitous for data analytics in the field of bioinformatics since the last decade. In the literature, more than 200 workflow systems [5] can be found, targeting specific scientific domains, different execution models and usability approaches. Workflow systems can be classified according to the model used to define the tasks and the data dependencies and to the characteristics

DOI 10.1109/eScience55777.2022.00049

of the engine that executes the workflow on the computing platform. With relation to the tasks definition features, some frameworks allow to explicitly define the workflow through a recipe file or a graphical interface while others permit the users to program their applications and let the runtime build a dependency graph from the user code. Another relevant characteristic for the classification of these frameworks is the level of integration with the different computing platforms as distributed environments (such as grids, clouds, and clusters), and HPC systems with multi-core architectures and accelerators (such as GPGPUs).

Amongst all these tools, in this paper we focus on the features that are more convenient for the orchestration of molecular dynamics simulations, taking into account interoperability across a variety of software and hardware environments, scalability, and reproducibility. In particular, we consider HPC-focused workflow managers that can compose and run workflows with advanced features as elasticity, adaptability, and fault tolerance.

Taverna [6], [7], Kepler [8], [9], Galaxy [10], [11] are well known graphical environments for the composition of workflows that can be stored and shared with other users of the community. These graphical environments facilitate the design of simple workflows, but the implementation of complex dynamic algorithms is difficult.

Other frameworks provide more generic interfaces to express the components of the workflow. Toil [12] is a Python workflow engine focused on the execution of pipelines. Pipelines are defined as jobs that can contain children jobs and follow-on jobs to explicitly force the synchronization of the execution. Nextflow [13], [14] provides a fluent DSL to implement and deploy scientific workflows and allows the adaptation of pipelines written in the most common scripting languages.

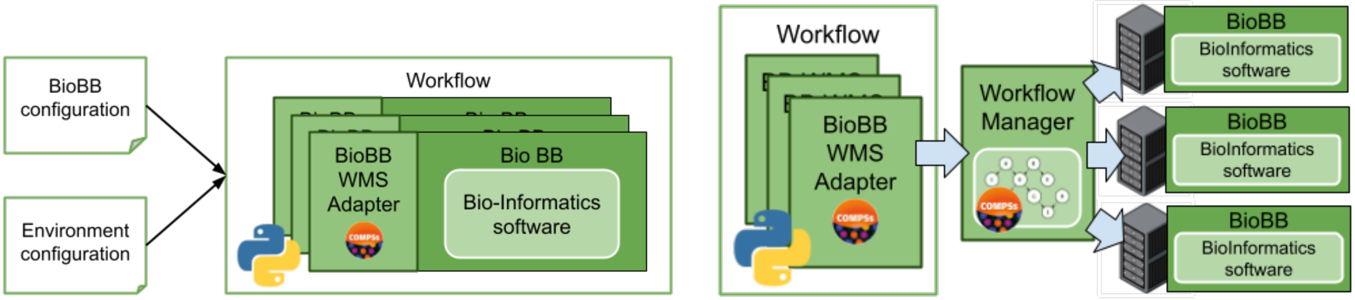
More bioinformatics specific environments have been recently developed. Crossbow [15] is a Python-based toolkit for workflow construction and execution, aimed mainly at Crossbow clusters but more generally at distributed computing environments. It provides an easy entry to cloud-based computing for biomolecular simulation scientists. Crossbow shares many of its design aspects with Parsl [16]. It provides tools to wrap Python functions and external applications (e.g., legacy MD simulation codes), in such a way that they can be combined into workflows using a task-based paradigm. Crossbow uses Dask [17] Distributed as the task scheduling and execution layer.

RADICAL-Cybertools [18] enables the execution of ensemble-based applications on a variety of high performance computing infrastructures. An increasing number of scientific domains are adopting and benefiting from ensemble-based applications. Most notably, MD simulations are nowadays executed as many parallel jobs of ns-length simulations rather than a single, long, and very large MPI job. AdaptiveMD [19] is a Python package designed to create HPC-scale workflows (parallel tasks) for adaptive sampling of biomolecular MD simulations. AdaptiveMD is designed as a distributed application that can be launched from a laptop or directly on an HPC resource and asynchronously automates the workflow creation and execution. Multiple adaptive sampling algorithms are fully automated with minimal user input, while advanced users can easily make modifications to workflow parameters and logic through the Python API. Runtime adaptations include the use of interim data as task properties such as analysis types or parameters, and workload properties such as task count or convergence criteria. To provide robust workflow management, AdaptiveMD is also integrated with the RADICAL Cybertools stack, which significantly enhances the runtime error detection and correction functionality, but has a much higher installation and configuration overhead.

The solution described in this paper advances the mentioned approaches in the move to developing robust and scalable scientific workflow without the requirement of deep programming knowledge on the users. The adoption of the combination of BioExcel BioBB and PyCOMPSs provides powerful features which simplify the development and executions of complex bio molecular workflows combining several types of heterogeneous tasks running in parallel on thousands of computing cores. Graphical workflow systems like Galaxy and KNIME have generally limited support for using HPC and HTC compute infrastructure in combination with high-performance codes like GROMACS, while our solution provides a solid solution for the execution of applications on a lot of computing backends without the need of adapting the code to a specific one.

III. Methodology

Joining different biomolecular tools in a complex pipeline is not always straightforward. It requires tedious developments to enable the interoperability between the different biomolecular simulation and analysis tools. Moreover, the need to execute the pipelines on distributed systems increases the complexity of these developments. Figure 1 provides an overview of the methodology proposed by the BioExcel Center of Excellence to simplify the development of dynamic, scalable, reliable and portable computational biomolecular workflows for distributed computing infrastructures. With this methodology, workflows are developed as simple Python scripts using reusable and extensible modules. The inputs of these workflows are two configuration scripts: one to configure the different workflow modules to setup the biomolecular system to be evaluated (mutations, number of simulation steps, etc.) and another one to indicate the properties related to the specific computing infrastructure for a specific execution (number of cores per node, gpus, etc.). The modules used inside the workflow are composed of two layers: the first layer is provided by the BioExcel Building Blocks (BioBBs), a software library designed to tackle the interoperability problem thanks to a simple wrapping approach. BioBBs are a collection of small wrappers written in Python where each building block encapsulates software components and provides a well-defined interface for input, output, configuration, and provenance. A standardised syntax is used in all the building blocks, with



(a) Development

(b) Execution

Fig. 1: Methodology overview

each of the wrappers internally performing the necessary format conversions for input and output, and launching the tool which runs unaltered. With this design, a large set of biomolecular tools such as GROMACS [20] (MD), HADDOCK [21] (Docking) or PMX [22] (Free energy calculations) can be executed using an homogeneous syntax, also providing a uniform and stable interface with enough information to plug the components into interoperable workflows as simple Python scripts. To transparently integrate the BioBBs with Workflows Management System, we propose a BioBB WMS Adapter layer, which is implemented as a set of decorators which interacts with the management system that takes care of the execution of the workflows. This adaptor layer transforms the local calls to the BioBBs into remote asynchronous calls. In this work, we have used PyCOMPSs to implement the adaptation layer. PyCOMPSs provides a programming model and a runtime system which allows developers to easily convert a sequential Python script to parallel workflows for distributed computing environments hiding the complexity of the parallelization and of the execution management. Next paragraphs provide more details about how this methodology provides different functionalities.

A. *Programmability, reusability and portability*

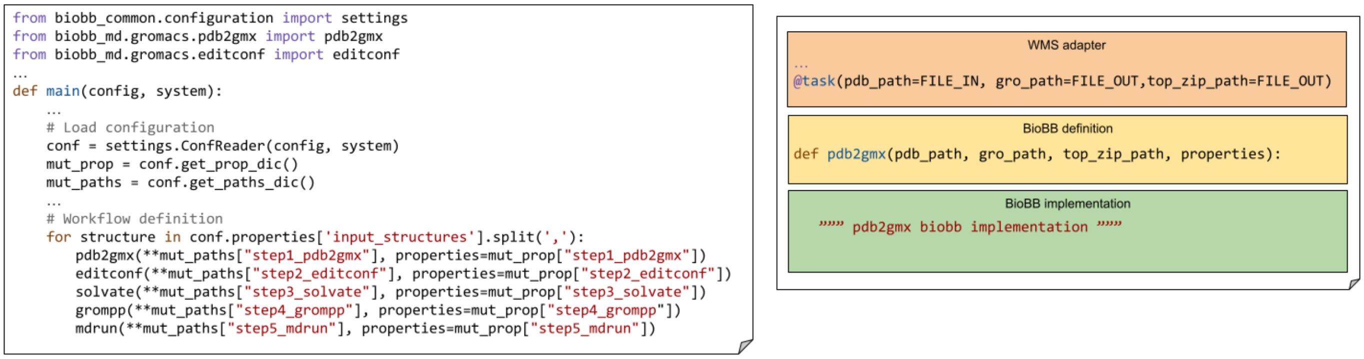
Figure 2 shows a code snippet to explain how workflows are developed with this methodology. On the left side of the figure, we can see a molecular dynamics setup workflow developed as a simple Python script where the executions of the computational biology tools are modelled as invocations to standard Python methods. These methods are provided by the BioBB libraries so developers do not need to implement them. They only need to import the modules of the tools they want to use in their workflows. In the right side of the picture, we can see a snippet of this BioBB module. The Python method definition is used as a common interface where the first parameters indicate required data inputs and outputs, and the properties parameter is a Python dictionary indicating the configuration of the execution of the tool. The body of the Python method contains the implementation while the decorators on top of the function are the PyCOMPSs annotations that enable the parallelization and execution of the BioBBs modules in the different computing nodes. All the BioBB methods are annotated with the task decorator which is also used to indicate the direction of the BioBB parameters. Every time a BioBB is invoked in a workflow, a task is created by the PyCOMPSs runtime that analyses the dependencies between the different BioBB according to the indicated direction. If the task is free of dependencies the BioBB execution will be scheduled and asynchronously executed in a computing resource. In this way, PyCOMPSs hides the complexity of the parallelization and the execution distribution of the workflow. Moreover, note that the workflow code is infrastructure agnostic, because the developer does not need to specify details of the infrastructure in the code that can be executed in different infrastructures without modifications.

B. *Multi-level parallelism*

Some computational biology tools are internally parallelized either to use different CPU cores in a single node (such as threads or OpenMP), across multiple nodes (such as MPI) or as a combination of both (MPI+openMP). It must be indicated in the workflow manager system in order to allocate the necessary resources for the execution. In our approach, this information is provided by adding decorators in the adapter part of the BioBB (@*multinode* and @*constraint*). Figure 3 shows a code example in the case of a GROMACS mdrun simulation. The *multinode* decorator indicates that the execution is using several nodes and the *constraint* decorator indicates the number of CPU or GPUs to use in each node of the execution. The values for these decorator are obtained from a set of environment variables defined in the environment configuration file of the workflow according to the computing platform capabilities. With this information, the PyCOMPSs runtime ensures the proper allocation of the required resources to perform the computation(via taskset, OMP_NUM_THREADS, CUDA_DEVICE,...)

C. *Reliability*

Computational biomolecular workflows are long lasting analyses that perform large amounts of simulations. Some of these simulations could fail or take too much time to get to a solution. However, the kind of analysis performed in these types of workflows can reach a solution even if there are simulations that fail. To manage this kind of features, PyCOMPSs



```
from biobb_common.configuration import settings
from biobb_md.gromacs.pdb2gmx import pdb2gmx
from biobb_md.gromacs.editconf import editconf
...
def main(config, system):
    ...
    # Load configuration
    conf = settings.ConfReader(config, system)
    mut_prop = conf.get_prop_dic()
    mut_paths = conf.get_paths_dic()
    ...
    # Workflow definition
    for structure in conf.properties['input_structures'].split(','):
        pdb2gmx(**mut_paths["step1_pdb2gmx"], properties=mut_prop["step1_pdb2gmx"])
        editconf(**mut_paths["step2_editconf"], properties=mut_prop["step2_editconf"])
        solvate(**mut_paths["step3_solvate"], properties=mut_prop["step3_solvate"])
        grompp(**mut_paths["step4_grompp"], properties=mut_prop["step4_grompp"])
        mdrun(**mut_paths["step5_mdrun"], properties=mut_prop["step5_mdrun"])
```

```
...
@task(pdb_path=FILE_IN, gro_path=FILE_OUT,top_zip_path=FILE_OUT)
```

```
def pdb2gmx(pdb_path, gro_path, top_zip_path, properties):
```

```
    """ pdb2gmx biobb implementation """
```

Fig. 2: Workflow implementation example. The left-hand side of the picture shows a code snippet of a workflow invoking different BioBB as standard python functions. The right-hand side of the picture shows how a BioBB and its WMS adapter is implemented.

```
@multinode(computing_nodes = '$TASK_NUM_NODES')
@constraint(processors=[
  {'processorType':'CPU','computingUnits':'$TASK_NUM_CPUS'},
  {'processorType':'GPU', 'computingUnits':'$TASK_NUM_GPUS'}
  ])
@task(...)
def mdrun(...):
    ...
```

Fig. 3: Decorators to indicate different levels of task parallelism.

provides additional properties in the task decorator to indicate how to proceed if the computation of this task fails or takes longer than expected [23]. These properties are included in the adaptor layer as shown in Figure 4. On the one hand, the *time_out* property indicates the maximum duration of the task. If a task is exceeding the indicated duration, its execution is cancelled and considered as a failure. On the other hand, the *on_failure* property indicates to the PyCOMPSs runtime what to do if a task of this type fails. In this case, the runtime will ignore the failure and the output data will be set to the specified default values (such as an empty file, None values,...)

```
@constraint(...)
@multinode(...)
@task(..., on_failure='IGNORE', time_out='$TASK_TIMEOUT')
def mdrun(...):
    ...
```

Fig. 4: Decorators to indicate reliability features.

Beside this functionality, we have also implemented a stop/restart mechanism to allow users to restart a workflow run without requiring to execute all the tasks again. This functionality is useful when executions exceed the wall clock limit of the queue system. To perform a safe stop, the runtime catches the signals sent by the queue system manager for cancelling the processes of the execution. When the signal is received, all the running and pending tasks are cancelled and the data generated by the already executed tasks is moved to its final location. When the execution is restarted and the workflow reaches the point to invoke a task, the adapter part of the BioBBs checks if the generated data is already available and that is not empty. If this data already exists, it skips the execution, otherwise, if it does not exist or it is empty it creates a task in the runtime to perform the missing computation.

D. Malleability

As stated in the introduction, the computing load of the workflows during the whole execution is not homogeneous; depending on the different phases, the workflow will use more or less computational resources. In task-based programming models like PyCOMPSs an application can be represented as a Direct-Acyclic-Graph (DAG) where nodes represent tasks and arrows data-dependencies between the tasks. This DAG also inherently stores information about the computational load required by the application at any point of the execution. The runtime can know which is the maximum achievable parallelism for a certain execution by analysing the generated graph and it can identify, by considering the available resources, whether the application has potential for further parallelism or it is under-utilising the current resources. For instance, when there are lots of dependency-free tasks pending for execution, the application could run faster by allocating more resources. However, when there are more resources than available ready tasks, we will under-utilise the system and some of the nodes will be idle or not running at the maximum capacity.

To overcome these situations, the runtime has an auto-scaling module which is able to scale up and down the computing resources used by the application according to its demands. During the application execution, the runtime generates profiling information about the previous tasks execution, including statistical information about the duration of each task. Combining this information with the task dependency graph, the PyCOMPSs runtime periodically estimates the remaining parallel workload (PW) as a sum of all the dependency-free task resource requirements ($R_{T_{ready_i}}$) multiplied by their mean execution time ($\overline{ET_{T_{ready_i}}}$).

$$PW = \sum_{\forall T_{ready}} R_{T_{ready_i}} \overline{ET_{T_{ready_i}}} \tag{1}$$

In the same way, the runtime can estimate the current infrastructure capacity (IC) with the sum of all resource capabilities ($C_{Resource_i}$) multiplied by the mean execution time to get a new resource from the infrastructure provider($\overline{RT}$).

$$IC = \sum_{\forall Resource} C_{Resource_i}\overline{RT} \tag{2}$$

These metrics are useful to determine when to add or remove a resource. If the estimated parallel workload is higher than the infrastructure capacity during the time to create a resource, it indicates that the application has enough load to be sped-up with a new resource. On the contrary, if it is smaller, it means that the application is starting to under-use resources.

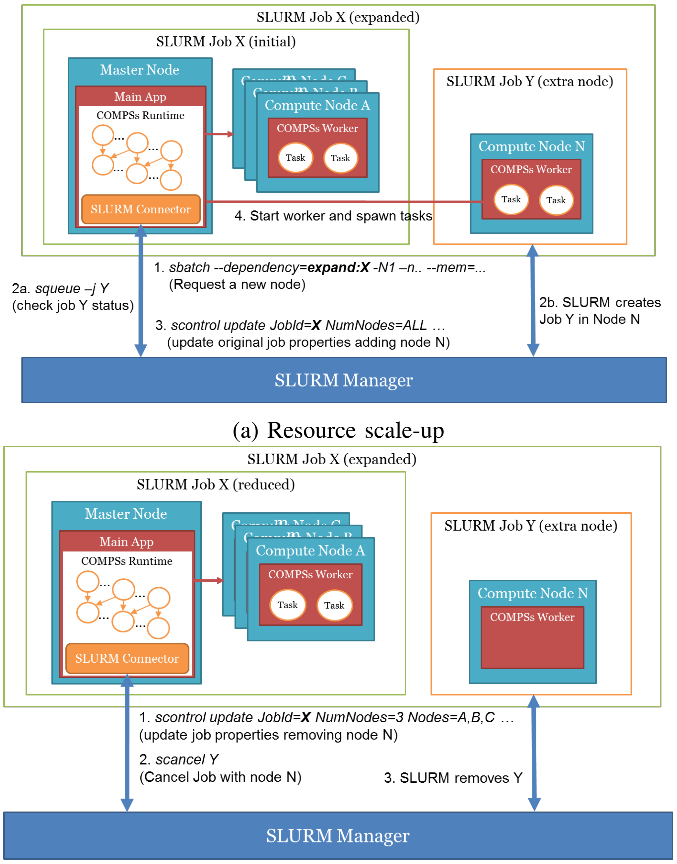
If the first situation is produced, the runtime contacts the infrastructure resource manager to request a new computational node and once the new resource is available, the runtime starts a worker process which spawns the execution of tasks in the new resource.

If the second situation is produced, the runtime has to decide which compute node is the best candidate to be removed. To do so, the runtime calculates the number of underused resources as the difference of the parallel load and the current infrastructure capacity. Then, it ranks the computing nodes by the capacity and the current running load (number of running tasks and estimated time to finish). Based on this rank, the runtime selects the node which contains the underused resources and which has the minimum running load. Once a node is selected, the runtime removes the node from the worker pool and once all the running tasks have finished, it contacts the infrastructure resource manager to release it.

In previous versions of PyCOMPSs [24], the auto-scaling features were applied to scale service executions in cloud environments, where the runtime contacts the resource provider API to create and destroy virtual machines. In this case, we have extended to auto-scale scientific workflows in clusters. Figure 5 illustrates how the runtime interacts with the SLURM resource manager to achieve the auto-scaling feature in clusters. To scale-up resources, the runtime requests an extra node by submitting a new job with the *sbatch* command indicating with the dependency flag that this job expands the main application job. This job will inherit the same QoS from the expanded job and will execute a COMPSs worker process in the requested extra node. The runtime detects when the job is running by polling SLURM with the *squeue* command. Once it is ready, the allocation of the new job is updated with the new resource with the *scontrol update job* command. In the case of scaling-down, the runtime contacts the SLURM manager to cancel the job which contains the node to remove (with the *scancel* command) and updates the main application job with the *scontrol update job* command.

IV. Workflows

To validate the functionalities like infrastructure agnosticity, scalability, resiliency, and malleability, we have implemented several workflows following the proposed methodology [25]. These workflows have been designed within the BioExcel CoE and applied to scientific use cases [26] where we have evaluated different molecular systems. These workflows and systems are briefly introduced in this section.



(a) Resource scale-up

(b) Resource scale-down

Fig. 5: Diagram about how the COMPSs runtime interacts with SLURM managed clusters to implement the resource elasticity in the workflow execution.

A. *Mutations MD Setup Workflow*

The Mutations MD setup workflow is an automated protocol to model residue mutations in 3D protein structures detected from genomics data, and prepare and run MD simulations for all the generated structures. The pipeline receives a PDB file (wild type protein 3D structure) and a set of mutations as input. Next, it prepares and runs MD simulations for each of the systems, thus obtaining static information (an ensemble of modelled structures for each of the protein variants), and dynamic data (trajectories for each of the protein variants). Both types of information can later be used in a comparative study. The workflow flowchart is represented in Figure 6. The structure of the workflow makes it a perfect case to study parallel work distribution, with a variable number of independent MD simulations to run, depending on the input number of mutations to model. Besides, the main tool used in the workflow (MD simulations) is implemented in various programming schemes, including GPU cards and openMP/MPI

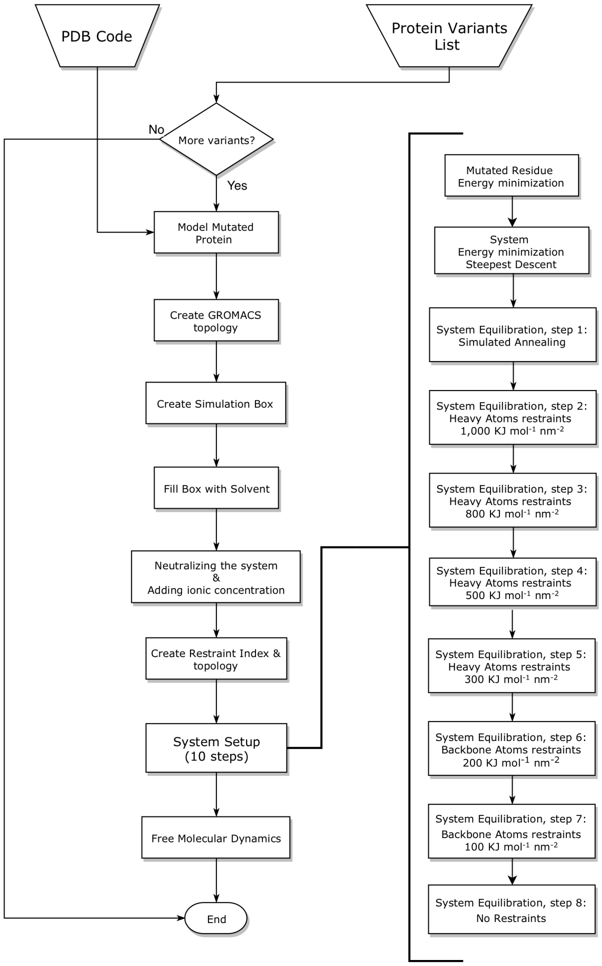

PDB Code
Protein Variants List
More variants?
No
Yes
Model Mutated Protein
Create GROMACS topology
Create Simulation Box
Fill Box with Solvent
Neutralizing the system & Adding ionic concentration
Create Restraint Index & topology
System Setup (10 steps)
Free Molecular Dynamics
End
Mutated Residue Energy minimization
System Energy minimization Steepest Descent
System Equilibration, step 1: Simulated Annealing
System Equilibration, step 2: Heavy Atoms restraints 1,000 KJ mol^{-1} nm^{-2}
System Equilibration, step 3: Heavy Atoms restraints 800 KJ mol^{-1} nm^{-2}
System Equilibration, step 4: Heavy Atoms restraints 500 KJ mol^{-1} nm^{-2}
System Equilibration, step 5: Heavy Atoms restraints 300 KJ mol^{-1} nm^{-2}
System Equilibration, step 6: Backbone Atoms restraints 200 KJ mol^{-1} nm^{-2}
System Equilibration, step 7: Backbone Atoms restraints 100 KJ mol^{-1} nm^{-2}
System Equilibration, step 8: No Restraints

Fig. 6: Mutation MD Setup flowchart

regimes, which makes the workflow also a perfect case to study the capacity to deal with different hardware architectures and parallelism levels.

B. Binding Affinity Workflow

The Binding Affinity workflow is an automated protocol to evaluate changes on binding affinity between a protein and its receptor upon residue mutations. The workflow uses the PMX tool [22] to generate alchemical residues, and GROMACS [20] to run a large series of short MD simulations required (thermodynamic integration, TI). The workflow flowchart is represented in Figure 7. Briefly, the method starts with two trajectories (wild type, mutated) coming from equilibrium MD simulations. From these trajectories, two different ensembles of structures are extracted and used to prepare and run alchemical perturbations from the wild type protein to the mutated protein (forward) and from the mutated protein to the wild type protein (reverse). The final output of the workflow are two histograms built with the results of the forward and reverse simulations. The intersection between the two histograms defines the final ΔG of binding.

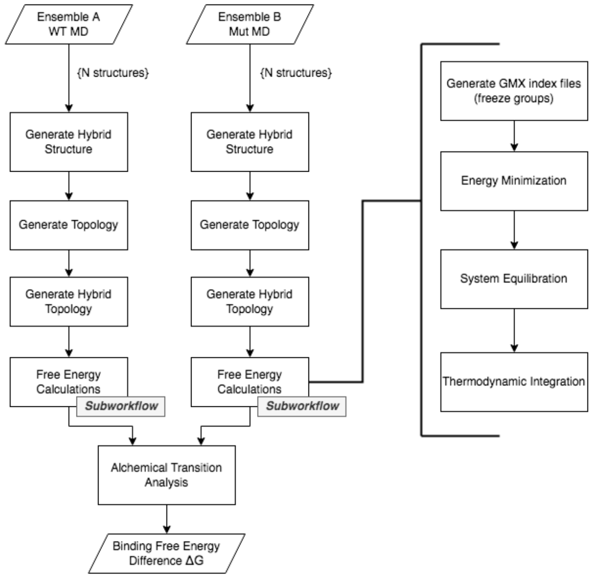


Fig. 7: Binding Affinity flowchart

The structure of the workflow makes it a perfect case to study PyCOMPSs fault tolerance, with a large number of complex independent workflow branches, with a certain probability of failure. In a typical run, 500 structures from each input ensemble are used to compute thermodynamic integrations, summing up to 1000 independent sub-workflows. The probability of failure for some of the workflow branches overall is high, but, unless the number of branches failing is high, it is not affecting the final value. Furthermore, this pipeline needs a considerable amount of computational resources and time, with the consequent risk of job cancelling, due to wall clock time or infrastructure downtime. Here the resilience properties to ensure the computation can be restarted and malleability properties to speed up the computation using more resources if available are very convenient.

C. Molecular Systems

Systems used in this study were chosen due to their interest in pharmaceutical research, and are briefly presented in this section.

The first system is the duodecimal peptide PMI, known to compete with p53 for binding to MDM2 or MDMX. p53 is critical for maintaining genetic stability and preventing cancer. MDM2 (E3 ubiquitin ligase) and its homologue MDMX act as negative regulators of p53. Designing inhibitors of MDM2 or MDMX is an attractive strategy for enhancing p53 activity and thus achieving the desired antitumoral therapeutic effect. The affinity of the peptide PMI is roughly two orders of magnitude higher than that of the same length p53 peptide. High-resolution crystal structures of both proteins with PMI are available (PDB identifiers: 3EQS and 3EQY, respectively). Kd dissociation constants of PMI and p53 for MDM2 and MDMX have been experimentally derived, and an additional Alanine scanning is also available for these proteins. All this experimental information can be used to test our methods to theoretically predict binding affinity.

The second system is the well-known SARS-Cov-2 Spike protein, and in particular, the molecules involved in the COVID-19 infection mechanism, the Spike Receptor Binding Domain (RBD) and the Human ACE2 (hACE2) receptor. High-resolution crystal structures of the protein complex are available (PDB identifier: 6VW1). Genetic changes in SARS-CoV-2 variants (α, β, γ, δ, o) are translated mainly into single amino acid mutations in the RBD domain of the viral Spike protein. The impact of these genetic changes can be determined using free energy calculations on the binding of RBD to the host receptor proteins. The system was chosen for its importance in the field, but also for its considerable complexity, both in size (65000 atoms) and in the number of charged (ionizable) amino acids.

V. Validation

This section describes the experiments performed to validate the described capabilities and the analysis of the results. These experiments consist of the execution of the implemented workflows configurations in a set of supercomputers from the PRACE partnership and Spanish Supercomputing Network (RES). The description of these clusters are described in Table I

TABLE I: Supercomputers description

Supercomputer	Computing Node Description
MareNostrum (MN)	48 core Intel Skylake CPU
Minotauro (MT)	16 core Intel Haswell CPU + 4 Nvidia K80 GPU
Tirant	16 core Intel Sandybridge CPU
Discoverer	128 core AMD EPYC 7H12 CPU

A. Portability

To validate the portability features, we have executed the Mutations MD setup workflow with the MDM2-PMI Alanine Scanning system which executes 8 Molecular Dynamic pipelines using GROMACS. It has been executed in three supercomputers: MareNostrum (MN), Tirant, and MinoTauro (MT). Table II shows the results for the different executions. The first two columns show the selected supercomputer and the number of nodes used in each execution and the specified constraints for the MD task. Then, the third column shows the elapsed time to execute the whole workflow, and the last column shows the performance reported by GROMACS to compute the MD simulation. In this table we can observe that the workflow execution is adapted to the computing infrastructure reaching the expected GROMACS performance according to the available computing resources.

TABLE II: Execution time in different supercomputers

Machine	MD Task Conf.	Exec. Time	GROMACS perf.
MN (8 Nodes)	48 CPU cores	97.5 min.	162.357 ns/day
MN (16 Nodes)	96 CPU cores	64.0 min.	250.787 ns/day
Tirant (8 Nodes)	16 CPU cores	372.8 min.	39.546 ns/day
MT (2 Nodes)	4 CPU + 1 GPU	197.5 min.	77.650 ns/day

B. Scalability

To validate the scalability of the methodology, we have performed a strong and weak scaling analysis for the Binding Affinity workflow using the SARS-Cov-2 Spike protein system in the Dicoverer supercomputer. Figure 8 shows the results of the strong scaling experiment. In this experiment, the workflow has been configured to evaluate 512 structures for each forward and reverse ensembles. The same workflow configuration has been executed with a different number of resources using a Discoverer node (128 cores) for each simulation. For each workflow execution, we have measured the execution time of the workflow computation discarding the runtime initilization and finalization, and we have computed the speed-up. The results show a good scalability up to 512 nodes (65,536 cores). It is close to the ideal until 256 nodes and reaches a speed-up of 311 for 512 nodes.

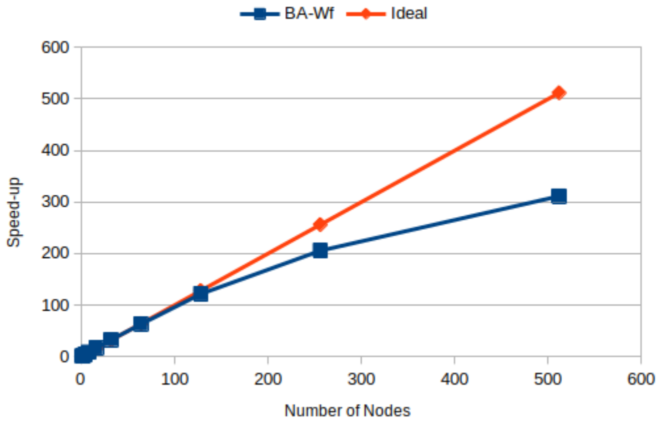


Fig. 8: Strong scaling analysis for Binding Affinity Workflow. (The baseline is the execution time using a single node)

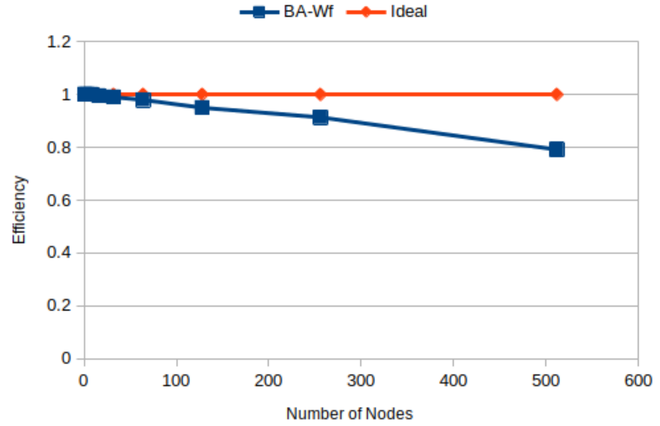


Fig. 9: Weak Scaling analysis for the Binding Affinity. (The baseline is the execution using a single node)

To complete the scalability analysis, we have designed a weak scaling experiment. In this case, we have used the same

Binding Affinity workflow and the same system (SARS-Cov-2 spike protein) but the number of structures to evaluate is increased by the same scale as the resources used for the execution, expecting to have a constant execution time. We have measured the execution time and we have computed the efficiency. The results of this experiment are presented in Figure 9 where we can see the efficiency is degraded with the amount of resources. For 512 nodes, the largest number of nodes, we have an 80% of efficiency. The speed-up and efficiency are diverging from the ideal case for two reasons: first, the workflow management overheads, such as task scheduling and data transfers, are growing with the number of tasks and nodes; and second, the Binding-affinity workflow is not completely embarrassingly parallel. Figure 10 shows an the generated graph for a simple case for 2 structures per ensemble. In this graph, we can observe that there is an initial phase (blue and white tasks) with a very limited parallelism (2 parallel tasks) and a the last phase whose execution is sequential.

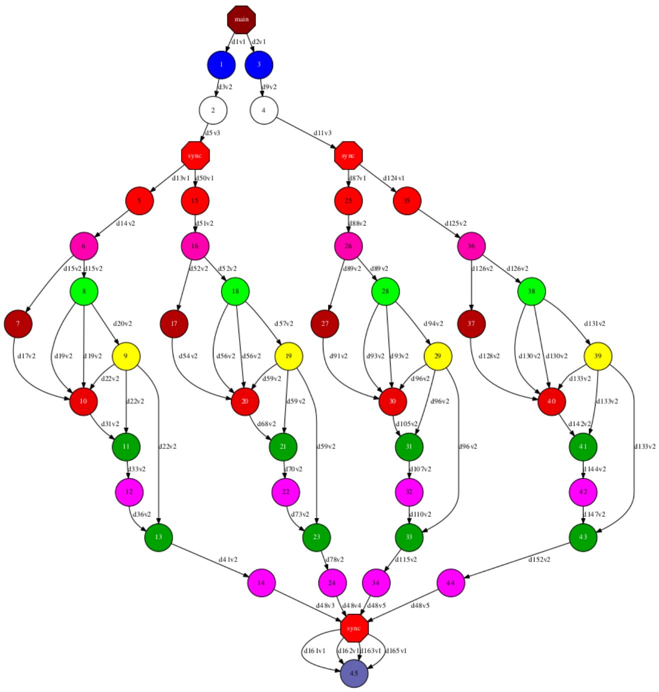


Fig. 10: Generated task graph for 2 structures per ensemble. Nodes in the graph represents tasks identified by the Workflow Management System and arrows represent the detected data dependencies

C. Reliability

To validate the reliability features, we have executed the Binding Affinity workflow in the MareNostrum supercomputer. In this case, we have included the evaluation of some of the structures that produce failures in the simulations. Figure 11 shows the plot of two analyses performed by the workflow. The left-side of these plots contains a representation of the values obtained for the simulations performed with the different structures for the forward (green) and backward (blue) ensembles. Note that there are some empty values which correspond to the failed simulations (red circles). According to the policy indicated in the *on_failure* property, these failures have been ignored and empty values have been set, allowing the finalisation of the workflow execution without altering the results of the analysis.

D. Malleability

To validate the malleability features of the proposed methodology, we have executed the Binding affinity workflow in the MareNostrum supercomputer with a malleable configuration, with 6 static computing nodes and 7 elastic computing nodes. Figure 12 shows the workload estimated by the runtime (blue) during the execution of the workflow, and the used amount of resources (red). During the first phase where setup tasks are quite small, the runtime decides to only use the static nodes. In the second phase where only the long simulations are pending the workload is increasing, so the runtime requests extra computing nodes that are released at the end of the execution, where the pending tasks can be finished with the static resources.

VI. Conclusions

The paper has presented a methodology to reduce the gap between biomolecular research and the high performance computing world. The motivation for this work comes from the analysis, performed in the context of the BioExcel project, of the current situation around the execution of biomolecular workflows in supercomputing facilities, and it has developed around two pillars: usability and efficiency. The methodology consists of the combination of two main components: First, a python library of platform agnostic building blocks for molecular dynamics (BioBB) has been used to address the usability requirement. A wide variety of complex pipelines can be developed as simple python scripts. The second component of the methodology is the PyCOMPSs task based programming model. It allows to convert python scripts in parallel workflows whose execution is distributed in different computing nodes with a minimal impact on the code (just requiring to add simple annotation on top of the BioBB method definitions).

We have described how the proposed methodology supports different types of simulations and adapts its execution to the available computing resources as well as other features like reliability and malleability. The validation of these features have been performed with two workflows implemented using the proposed methodology. These workflows have been executed in different premises with different processors to see how the execution is making use of the available hardware in each case. We have also verified that the workflow execution is finished even when some tasks are failing and the runtime system can adapt the infrastructure to the workload generated by the workflow execution. Finally, a scalability analysis has been performed executing the workflow with up to 65,536 cores, demonstrating that the workflows developed with this methodology can be easily scaled to a large number of nodes

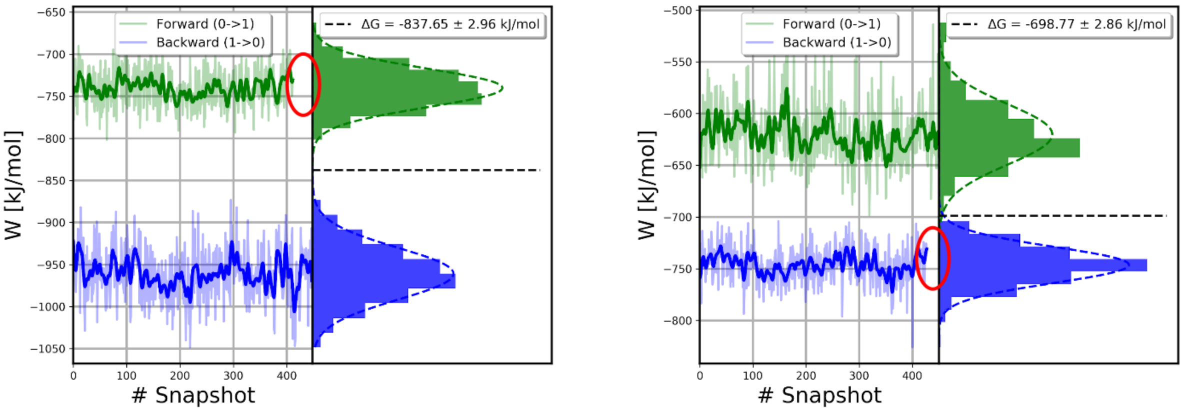


Fig. 11: Binding Affinity workflow results with failures. The red circle indicates the simulations with empty results which are produced by the failures in the workflow

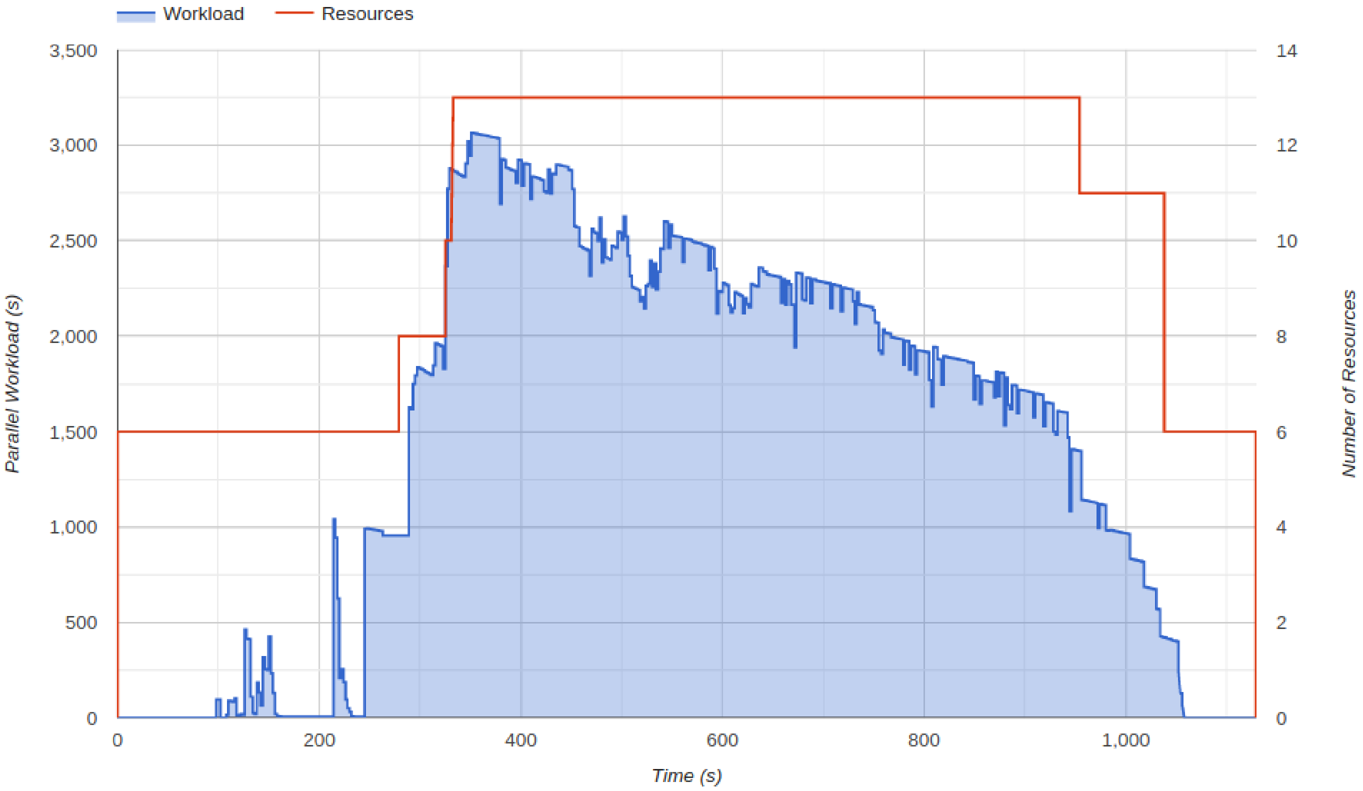


Fig. 12: Malleability with the Binding Affinity workflow. Blue graph shows the parallel workload estimated by the runtime and the red line shows the number of resources assigned to the workflow execution.

and that the optimal configuration of the execution parameters can be obtained without modifying the user code.

Regarding future work, we will work on simplifying the development of more complex workflows and extending the BioBB library with the introduction of data analytics in the workflows. In particular, we will investigate how to couple the adoption of High Performance Data Analytics (HPDA) with High Performance Computing (HPC) techniques. The use of HPC parallel processing to run powerful data analysis software tools opens the possibility to examine massive datasets within a reasonable time. A set of HPDA building blocks will be developed, starting with a recently developed library for distributed computing integrated on top of the PyCOMPSs framework and focused on machine learning [27].

ACKNOWLEDGMENT

This work has been supported by Spanish Ministry of Science and Innovation MCIN/AEI/10.13039/501100011033 under contract PID2019-107255GB-C21, by the Generalitat de Catalunya under contracts 2017-SGR-01414 and 2017-SGR-1110, by the European Commission through the BioExcel Center of Excellence (Horizon 2020 Framework program) under contracts 823830, and 675728. This work is also partially supported by the CECH project which has been co-funded with 50% by the European Regional Development Fund under the framework of the ERFD Operative Programme for Catalunya 2014-2020, with a grant of 1.527.637,88€

REFERENCES

[1] BioExcel Center of Excellence. (2018) BioExcel building blocks. [Online]. Available: https://github.com/bioexcel/biobb

[2] R. M. Badia and et al., "COMP superscalar, an interoperable programming framework," *SoftwareX*, vol. 3, pp. 32–36, 12 2015. [Online]. Available: https://doi.org/10.1016/j.softx.2015.10.004

[3] E. Tejedor, R. M. Badia, J. Labarta, and et al., "PyCOMPSs: Parallel computational workflows in Python," *The International Journal of High Performance Computing Applications (IJHPCA)*, vol. 31, pp. 66–82, 2017. [Online]. Available: http://dx.doi.org/10.1177/1094342015594678

[4] R. Amela, C. Ramon-Cortes, J. Ejarque, J. Conejero, and R. M. Badia, "Enabling python to execute efficiently in heterogeneous distributed infrastructures with pycompss," in *Proceedings of the 7th Workshop on Python for High-Performance and Scientific Computing*, 2017, pp. 1–10.

[5] Common Workflow Language Group. (2019) Existing Workflow systems. [Online]. Available: https://github.com/common-workflow-language/common-workflow-language/wiki/Existing-Workflow-systems

[6] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. Nieva de la Hidalga, M. P. Balcazar Vargas, S. Sufi, and C. Goble, "The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud," *Nucleic Acids Research*, vol. 41, no. W1, pp. W557–W561, 05 2013. [Online]. Available: https://doi.org/10.1093/nar/gkt328

[7] Taverna Committers. (2014) Apache Taverna. [Online]. Available: https://taverna.incubator.apache.org/

[8] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the Kepler system," *Concurrency and Computation: Practice and Experience*, vol. 18, no. 10, pp. 1039–1065, 2006.

[9] UC Davis and UC Santa Barbara and UC San Diego. (2004) The Kepler Project. [Online]. Available: https://kepler-project.org/

[10] E. Afgan, D. Baker, M. van den Beek, D. Blankenberg, D. Bouvier, M. Čech, J. Chilton, D. Clements, N. Coraor, C. Eberhard, B. Grüning, A. Guerler, J. Hillman-Jackson, G. Von Kuster, E. Rasche, N. Soranzo, N. Turaga, J. Taylor, A. Nekrutenko, and J. Goecks, "The Galaxy platform for accessible, reproducible and collaborative biomedical analyses: 2016 update," *Nucleic Acids Res.*, vol. 44, no. W1, pp. W3–W10, 2016.

[11] Galaxy Team. (2005) Galaxy. [Online]. Available: https://usegalaxy.org/

[12] J. Vivian, A. A. Rao, F. A. Nothaft, C. Ketchum, J. Armstrong, A. Novak, J. Pfeil, J. Narkizian, A. D. Deran, A. Musselman-Brown, H. Schmidt, P. Amstutz, B. Craft, M. Goldman, K. Rosenbloom, M. Cline, B. O'Connor, M. Hanna, C. Birger, W. J. Kent, D. A. Patterson, A. D. Joseph, J. Zhu, S. Zaranek, G. Getz, D. Haussler, and B. Paten, "Toil enables reproducible, open source, big biomedical data analyses," *Nature Biotechnology*, vol. 35, no. 4, pp. 314–316, 2017. [Online]. Available: https://doi.org/10.1038/nbt.3772

[13] P. Di Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, and C. Notredame, "Nextflow enables reproducible computational workflows," *Nature biotechnology*, vol. 35, no. 4, p. 316, 2017.

[14] Barcelona Centre for Genomic Regulation. (2014) Nextflow: A DSL for parallel and scalable computational pipelines. [Online]. Available: https://www.nextflow.io/

[15] Nottingham's Physical and Theoretical Chemistry Group. (2019) Crossbow. [Online]. Available: https://github.com/ChrisSuess/Project-Xbow

[16] Y. Babuji, A. Woodard, Z. Li, D. S. Katz, B. Clifford, R. Kumar, L. Lacinski, R. Chard, J. M. Wozniak, I. Foster, M. Wilde, and K. Chard, "Parsl: Pervasive Parallel Programming in Python," in *Proceedings of the 28th International Symposium on High-Performance Parallel and Distributed Computing*, ser. HPDC '19. New York, NY, USA: ACM, 2019, pp. 25–36. [Online]. Available: http://doi.acm.org/10.1145/3307681.3325400

[17] *Dask: Library for dynamic task scheduling*, Dask Development Team, 2016. [Online]. Available: https://dask.org

[18] V. Balasubramanian, S. Jha, A. Merzky, and M. Turilli, "RADICAL-Cybertools: Middleware Building Blocks for Scalable Science," *ArXiv*, vol. abs/1904.03085, 2019.

[19] Oak Ridge National Laboratory. (2019) AdaptiveMD. [Online]. Available: https://github.com/markovmodel/adaptivemd

[20] S. Pronk, S. Páll, R. Schulz, P. Larsson, P. Bjelkmar, R. Apostolov, M. R. Shirts, J. C. Smith, P. M. Kasson, D. van der Spoel, B. Hess, and E. Lindahl, "GROMACS 4.5: a high-throughput and highly parallel open source molecular simulation toolkit," *Bioinformatics*, vol. 29, no. 7, pp. 845–854, 02 2013. [Online]. Available: https://doi.org/10.1093/bioinformatics/btt055

[21] C. Dominguez, R. Boelens, and A. M. Bonvin, "Haddock: a protein- protein docking approach based on biochemical or biophysical information," *Journal of the American Chemical Society*, vol. 125, no. 7, pp. 1731–1737, 2003. [Online]. Available: https://doi.org/10.1021/ja026939x

[22] V. Gapsys, S. Michielssens, D. Seeliger, and B. L. de Groot, "pmx: Automated protein structure and topology generation for alchemical perturbations," *Journal of Computational Chemistry*, vol. 36, no. 5, pp. 348–54, 2015.

[23] J. Ejarque, M. Bertran, J. Á. Cid-Fuentes, J. Conejero, and R. M. Badia, "Managing failures in task-based parallel workflows in distributed computing environments," in *European Conference on Parallel Processing*. Springer, 2020, pp. 411–425.

[24] F. Lordan, R. M. Badia, and et al., "ServiceSs: an interoperable programming framework for the Cloud," *Journal of Grid Computing*, vol. 12, no. 1, pp. 67–91, 3 2014. [Online]. Available: https://digital.csic.es/handle/10261/132141

[25] BioExcel Center of Excellence. (2021) BioExcel HPC Workflows. [Online]. Available: https://github.com/bioexcel/biobb_hpc_workflows/tree/condapack

[26] M. Wieczór, V. Genna, J. Aranda, R. M. Badia, J. L. Gelpí, V. Gapsys, B. L. de Groot, E. Lindahl, M. Municoy, A. Hospital, and M. Orozco, "Pre-exascale hpc approaches for molecular dynamics simulations. covid-19 research: A use case," *WIREs Computational Molecular Science*, vol. n/a, no. n/a, p. e1622, 2022. [Online]. Available: https://wires.onlinelibrary.wiley.com/doi/abs/10.1002/wcms.1622

[27] J. Álvarez Cid-Fuentes, S. Solà, P. Álvarez, A. Castro-Ginard, and R. M. Badia, "dislib: Large Scale High Performance Machine Learning in Python," in *Proceedings of the 15th International Conference on eScience*, 2019, pp. 96–105.