SIM-SITU: A Framework for the Faithful Simulation of in-situ Workflows

Valentin Honoré, Tu Mai Anh Do, Loïc Pottier, Rafael Ferreira da Silva, Ewa Deelman, Frédéric Suter

Sim-Situ: A Framework for the Faithful Simulation of *in-situ* Workflows

Valentin Honoré[1], Tu Mai Anh Do[2], Loïc Pottier[2],
Rafael Ferreira da Silva[3], Ewa Deelman[2], Frédéric Suter[1]

[1]IN2P3 Computing Center / CNRS, Lyon - Villeurbanne, France
valentin.honore@cc.in2p3.fr, fsuter@cc.in2p3.fr
[2]*USC Information Sciences Institute, Marina del Rey, CA, USA*
tudo@isi.edu, lpottier@isi.edu, deelman@isi.edu
[3]*Oak Ridge National Laboratory, Oak Ridge, TN, USA*
silvarf@ornl.gov

Abstract

The amount of data generated by numerical simulations in various scientific domains such as molecular dynamics, climate modeling, biology, or astrophysics, led to a fundamental redesign of application workflows. The throughput and the capacity of storage subsystems have not evolved as fast as the computing power in extreme-scale supercomputers. As a result, the classical post-hoc analysis of simulation outputs became highly inefficient. In-situ workflows have then emerged as a solution in which simulation and data analytics are intertwined through shared computing resources, thus lower latencies.

Determining the best *allocation*, i.e., how many resources to allocate to each component of an *in-situ* workflow; and *mapping*, i.e., where and at which frequency to run the data analytics component, is a complex task whose performance assessment is crucial to the efficient execution of *in-situ* workflows. However, such a performance evaluation of different allocation and mapping strategies usually relies either on directly running them on the targeted execution environments, which can rapidly become extremely time- and resource-consuming, or on resorting to the simulation of simplified models of the components of an *in-situ* workflow, which can lack of realism. In both cases, the validity of the performance evaluation is limited.

To address this issue, we introduce Sim-Situ, a framework for the faithful simulation of *in-situ* workflows. This framework builds on the SimGrid toolkit and benefits of several important features of this versatile simulation tool. We designed Sim-Situ to reflect the typical structure of *in-situ* workflows and thanks to its modular design, Sim-Situ has the necessary flexibility to easily and faithfully evaluate the behavior and performance of various allocation and mapping strategies for *in-situ* workflows. We illustrate the simulation capabilities of Sim-Situ on a Molecular Dynamics use case. We study the impact of different allocation and mapping strategies on performance and show how users can leverage Sim-Situ to determine interesting tradeoffs when designing their *in-situ* workflow.

1 Introduction

The tremendous volumes of data generated by numerical simulations in various scientific domains such as molecular dynamics, climate modeling, biology, or astrophysics, led to a fundamental redesign of application workflows. Due to the growing discrepancy between storage subsystems performance and computing power in extreme-scale supercomputers, moving large volumes of data from computational resources to disks may have a dramatic impact on performance [1]. This makes the classical post-hoc analysis of simulation outputs highly inefficient. To overcome these issues, *in-situ* workflows intertwine simulation and data analytics within shared computing resources. The objective is to take advantage of data locality by consuming partial simulation results to periodically run the data analysis. Then, the final outcome of the workflow execution will only be the result of the data analytics phase, which is usually much smaller than the entire simulation data.

A common challenge to the many software frameworks that have been developed to efficiently process *in-situ* workflows is to decide what is the best *allocation*, i.e., how many resources to allocate to each component of an *in-situ* workflow, and *mapping*—where and at which frequency run the data analytics component. This is a complex task whose performance assessment is crucial to the efficient execution of *in-situ* workflows. However, such a performance evaluation of different allocation and mapping strategies usually relies either on directly running them on the targeted execution environments or on resorting to the simulation of simplistic models of the components of an *in-situ* workflow. The former requires access to clusters and an important time to tune the framework with regard to the hardware and software available on the target machine and thus can rapidly become extremely time- and resource-consuming, while the latter can lack of realism. In both cases, the validity of the performance evaluation is usually limited to a narrow set of configurations.

In this paper, we present **Sim-Situ**, a framework for the faithful simulation of *in-situ* workflows. Sim-Situ builds on the popular SimGrid toolkit [2] and benefits of several key features of this versatile simulation framework. Indeed, SimGrid enables the simulation of large-scale distributed applications in a way that is accurate (via validated simulation models), scalable (ability to run large scale simulations on a single computer with low compute, memory, and energy footprints), and expressive (ability to simulate arbitrary platform, application, and execution scenarios). We designed Sim-Situ to reflect the typical structure of *in-situ* workflows with three distinct modules that respectively, (i) simulate the unmodified simulation component of an *in-situ* workflow; (ii) mimic the behavior of an underlying Data Transport Layer (DTL); and (iii) abstract the data analytics processing.

Thanks to this modular design, Sim-Situ has the necessary flexibility to easily and faithfully evaluate the behavior and performance of various allocation policies for *in-situ* workflows. We illustrate the simulation capabilities of Sim-Situ on a Molecular Dynamics use case. We study the impact of different allocation and mapping strategies on performance and show how users can leverage Sim-Situ to determine interesting tradeoffs when designing their *in-situ* workflow.

The remaining of this paper is organized as follows. Section 2 presents the related work. In Section 3, we describe the architecture of the Sim-Situ framework and detail its different features and advantages. In Section 4, we describe how we used Sim-Situ to simulate a Molecular Dynamics (MD) *in-situ* workflow. Thanks to this implementation, we evaluate different *in-situ* allocation and mapping strategies in Section 5. Finally, Section 6 summarizes our contributions and presents some future research directions.

2 Related Work

The evaluation of the performance of the execution of *in-situ* workflows, and in particular that of different allocation and mapping strategies for both the simulation and data analytics components, is a complex and multi-parametric problem. Different approaches have been proposed in the literature to ascertain the performance gains brought by an *in-situ* (or *in-transit*) execution of a given scientific workflow application and determine the best configuration deployment of its components on

a given target platform. We distinguish these approaches depending on whether they rely on actual experiments [3–7] or resort to simulation [8–10] to evaluate the performance of *in-situ* workflows. The former is intrinsically time- and resource-consuming while the latter may suffer from simplification biases when the abstract versions of the *in-situ* workflow components are developed.

To limit the cost of the experiments required to perform performance evaluations, some works focused on a particular component of the *in-situ* workflow–the Data Transport Layer (DTL) that connects the simulation and the analysis. In such cases, series of experiments are conducted either using the real application [3] or by leveraging data access traces to mimic an application behavior [6,7]. Another approach consisted in reducing the experimental configuration space to a selected set of promising configurations. For instance, Malakar *et al.* tackled the allocation and mapping of an *in-situ* workflow as an optimization problem expressed as a Mixed-Integer Linear Program [4, 5]. Solving this optimization problem results on a set of feasible *in-situ* analyses whose performance has been assessed through experimentation in an actual platform. Lorhmann *et al.* leveraged surrogate models (i.e., proxy-applications) of expensive numerical simulation codes [8] to abstract application models. While this approach substantially reduces the cost of the experiments, it still captures the most important features of the considered application.

Only a few recent attempts has leveraged simulation to enable the exploration of the *in-situ* parameter space. Aupy *et al.* designed a numerical simulator [9] that measures evaluation metrics for scheduling decisions by solving optimization problems on resource allocation and partitioning for an *in-situ* analysis set. The simulator used a predetermined set of simulation parameters to study the impact of the *in-situ* analyses that are scheduled on the performance of the entire *in-situ* execution. In a previous work [10], we extrapolated benchmarking performance of realistic MD simulation engines to create a synthetic MD simulation that emulates the behavior of MD simulations while using fewer computing resources. Our proposed synthetic MD simulation considered computation units as timing delays without actually performing any computing operations.

To the best of our knowledge, this is the first work that proposes a faithful and scalable simulation framework that models the behavior of all the components of *in-situ* workflows.

3 Sim-Situ Architecture

In this paper, we focus on a particular type of *in-situ* workflows whose structure is depicted in the top part of Figure 1. These workflows feature the following main components:

- A *Simulation* component that performs domain-specific computations, generally in an iterative process. This component periodically, i.e., after a predefined number of iterations, produces some scientific data;
- A *Data Analytics* component that consumes the data generated by the simulation component and applies one or several analysis kernels;
- A *Data Transport Layer* (DTL), whose complexity depends on the degree of coupling between the two former components and the resources they are allocated to, is responsible for efficient data transfers.

To faithfully simulate such *in-situ* workflows, we introduce the Sim-Situ framework that leverages several features of the SimGrid toolkit [2]. SimGrid is an open-source versatile framework for developing simulators of distributed applications that are executed on heterogeneous distributed platform. One of the key strengths of SimGrid is to not trade accuracy for scalability. Its fast simulation models have been theoretically and experimentally evaluated and validated [11] and make it possible to run large-scale simulations on a single machine. The way we simulate the components of an *in-situ* workflow is illustrated by the bottom part of Figure 1 and detailed hereafter.

Simulation Component. To maximize the realism of the simulation of a parallel application, such as the simulation component on an *in-situ* workflow, an appealing approach is to directly simulate the

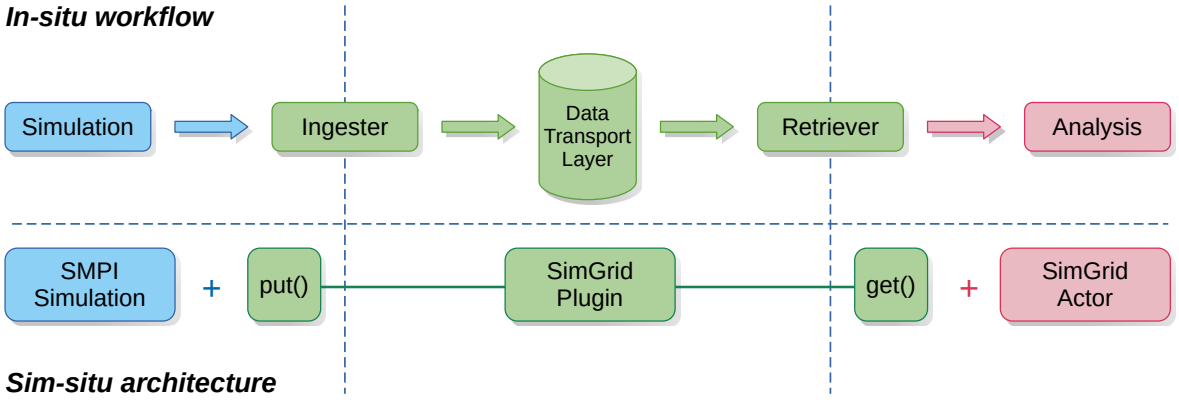


Figure 1: Software architectures of a generic *in-situ* workflow (top) and the SIM-SITU framework (bottom).

unmodified code of the parallel application itself. This ensures that the simulation not only captures the computations executed on the different processes but also the exact communication pattern of the application.

This approach is at the origin of SMPI, a flexible simulator of MPI applications [12] that comes with the SimGrid distribution. SMPI works seamlessly with unmodified MPI programs written in C, C++, or FORTRAN. As part of its integration testing effort[1] SMPI simulates the execution of multiple proxy- and full-scale applications and HPC runtimes. The code is compiled using one of the language specific compilers shipped with SimGrid. For instance, assuming the program is written in C, one simply compiles it with `smpicc`, instead of `mpicc`, and executes it with `smpirun`, instead of `mpirun`. The only difference is that `smpirun` takes one extra command-line argument, `-platform`, which allows the user to describe the hardware platform on which the execution of the MPI program is to be simulated. This platform description specifies a network topology between compute nodes, where network links have specified latencies and bandwidths, and compute nodes have specified compute speeds, and numbers of cores.

The basic principle behind SMPI simulations is as follows. The code of the MPI program is executed as is, but the MPI ranks actually execute as threads in a single process, and thus share the same address space. Each time an MPI function is called, control is handed off to SMPI where network operations are replaced by simulated delays. These delays are computed using the simulation models at the core of SimGrid [13]. Each block of code in between two MPI calls is benchmarked on the machine used to execute the simulation. This is possible because, with SMPI, MPI ranks execute as threads in mutual exclusion. These benchmarked execution times can then be scaled and simulated as compute delays that correspond to the compute speeds of the nodes in the simulated platform. In this way, SMPI simulates both communication and computation operations as computed delays.

Data Analytics. The data analytics component of an *in-situ* workflow is usually specific to a given scientific study. Moreover, its complexity can vary greatly depending on what knowledge scientists want to extract from the simulation data. It can range from a simple computation of some performance metrics to help steer the simulation to a complex visualization of the current state of the simulation. Then, to simulate this component in SIM-SITU we decide to simply abstract its execution time within one or several SimGrid *actors*. A SimGrid actor corresponds to a simulated process that can consume some simulated *resources* (e.g., CPU time, network bandwidth, or storage space) by performing some simulated *activities* (e.g., executing a computation, doing some I/O, or communicating with another actor).

Isolating and abstracting the data analytics component within actors out of the MPI world offers

[1] https://framagit.org/simgrid/SMPI-proxy-apps

a great flexibility to Sim-Situ and let us envisage multiple scenarios. First, decoupling the simulation from the analysis in Sim-Situ's code allows us to easily decide where to map these actors. Indeed, an actor can be started on any node of the simulated cluster, that execution location being specified at the creation of the actor, or predefined in the description of an initial static deployment of the different actors. Second, abstracting the analysis as an activity consuming CPU resources allows us to easily study various simulation over analysis time ratios and determine what should be the best allocation and mapping strategy in each scenario. Last, it is also possible to spawn new actors, stop existing ones, or even migrate them from one node to another while the SimGrid simulation is running. This would allow Sim-Situ to study complex scenarios where the analysis load evolves along time.

Data Transport Layer. To simulate data exchanges between the simulation and the data analytics components, we opted for the development of a SimGrid plugin. A SimGrid plugin is a standalone piece of code that composes some of basic concepts exposed by the SimGrid API to offer a higher level of abstraction useful in a specific context. In this paper, the proposed plugin relies on data structures accessed through a Producer-Consumer synchronization mechanism commonly used in actual DTLs [14, 15]. Using this plugin only requires to include an extra header file in the application code to use the functions it provides.

Two features of this plugin are especially interesting in the context of Sim-Situ. First, it offers two different internal implementations of the message queue that allow us to consider three different communication modes between the main components of an *in-situ* workflow. The former is a standard *queue* (of configurable size). In this case, data exchanges are instantaneous, i.e., do not induce any advance of the simulated clock, but respect the flow dependencies between the producers and consumers. This allows us to study scenarios where the analysis component has a direct and seamless access to the data produced by the simulation component, or to only focus on the computational element of the *in-situ* workflow and study the impact of different allocation and mapping schemes on that element alone.

The latter implementation leverages the concept of *mailbox* used by SimGrid to implement inter-process communications. It acts as a rendez-vous point between a sender and a receiver processes. When both meet on that rendez-vous point, the actual communication starts. SimGrid mailboxes use a queue to store unmatched communications, i.e., when one side is waiting for the other, which ensures the respect of flow dependencies. An interesting feature of mailboxes is that depending on where the processes are located, we can simulate two different ways to exchange data. If the producer and the consumer are located on the same compute node, and thus sharing a memory space, the simulated communication will go through the loopback of this node. This allows us to simulate a memory copy of data between the simulation and data analytics components. Conversely, if the producer and the consumer are located on different nodes, the communication will go over the interconnection network. This makes it possible to easily compare *in-situ* and *in-transit* scenarios.

The second interesting feature of the proposed plugin is that accesses to the simulated DTL can be done either in synchronous or asynchronous modes. Again, this offers more flexibility and broaden the range of scenarios that can be studied with Sim-Situ.

4 Application to MD In-situ Workflows

To assess the accuracy and illustrate the flexibility of the proposed Sim-Situ framework, we consider the simulation of a Molecular Dynamics (MD) *in-situ* workflow. MD aims at studying the evolution of molecular systems at the atomic scale and, is one of the most prominent types of numerical simulations currently running on extreme-scale systems.

Application. More precisely, we relied for our experiments on the ExaMiniMD proxy-application [16, 17] which is part of the Exascale Computing Project Proxy App Suite v4.0 [18]. ExaMiniMD captures both the computation and communication schemes that are implemented in the classical MD code LAMMPS [19]. As other proxy-applications, ExaMiniMD shows a good balance between having a

compact and manageable code and representing the main performance concerns of MD applications. ExaMiniMD belongs to the family of the all-atom MD simulations. It computes floating-point intensive pairwise atom-atom unbounded interactions over a certain period of time. The simulated system corresponds to a set of particles distributed in a 3D volume. The main loop computes the trajectories of the particles according to a Verlet time integration method and the short-range forces between particles as a Lennard-Jones potential. The parallelization of this MD problem follows a typical domain decomposition approach. Each MPI rank manages a sub-volume and a halo to exchange with its neighbors periodically. The periodicity of these exchanges can be set to `X` by adding `neigh_modify every X` to the input file. All the data exchanges in the simulation component rely on point-to-point MPI communications with asynchronous receives.

The data analytics component of ExaMiniMD consists in periodically computing the temperature, potential energy, and kinetic energy of the system. It is entangled with the simulation code and uses the same process mapping. Each MPI rank locally computes the different metrics and then enters a global `MPI_Allreduce` function that leads to the final value. The periodicity of this analysis is set by the `thermo` input parameter. If no value is given, the computation of these values are only done before and after the main simulation loop.

Experimental Platform. Our set of experiments and simulations were conducted on the *dahu* cluster of the Grid'5000 experimental testbed. This cluster consists of 32 nodes that comprise two Intel Xeon Gold 6130 CPUs with 16 cores each and 192 GiB of memory. These nodes are interconnected through a 10 Gb/s Ethernet network. We leverage an existing thorough calibration of the SMPI network model for this same cluster [20] to ensure our simulated results are accurate. This calibration runs a series of tests on a limited number of nodes to assess the performance of point-to-point communications and saturate a switch. It can then be used to extrapolate the size of a given cluster beyond its actual number of nodes.

We used the git version of ExaMiniMD, compiled with g++ v8.3.0 and linked to Kokkos v3.3.01 and OpenMPI v3.1.3. We used the `Serial` Kokkos device with one MPI rank per core as we observed a performance degradation with the `Pthread` and `OpenMP` devices combined with MPI. For the simulation of ExaMiniMD, we relied on SimGrid v3.29.

4.1 Simulation Component

In this section, we analyze the performance and accuracy of the simulation with SMPI of the unmodified code of the ExaMiniMD application. Enabling this simulation only requires minimal modification to the `Makefile` file to indicate that the compiler to use is `smpicxx`.

An additional and optional modification of the code of ExaMiniMD can be made to drastically reduce the simulation time. SMPI offers to replace time-consuming computational parts of the simulated application by delays to speedup the simulation. These delays are estimated by sampling the execution time of a given kernel or loop body either for a predefined number of times or until the standard deviation of the samples is under a given threshold. All the subsequent calls are then replaced by the average execution time of these samples. Finally, this sampling can be done either at a *local* scale, i.e., each MPI rank determines its own delay from the samples it executed, or at a *global* scale, i.e., the delay is determined from samples executed by all the MPI ranks.

We used this feature of SMPI on the most time-consuming kernel of ExaMiniMD which is the call to `ForceLJNeigh::compute`. This particular compute-bound kernel represents 69% of the execution time of the application [21]. This corresponds to a 1-line modification of the code to call the SMPI sampling macro with its parameters. We chose to run 150 samples with a standard deviation threshold of 0.002 in our experiments.

Characterization. Figure 2 compares the time needed to run 8,000 iterations of a 3D Lennard-Jones melt on a $70 \times 70 \times 70$ region with ExaMiniMD and the time needed to simulate this execution with SMPI for different numbers of MPI ranks. Data exchanges among ranks occur every 20 iterations and

the analytics phase is triggered every 50 iterations. We map an MPI rank per core and use a single Kokkos thread per MPI rank.

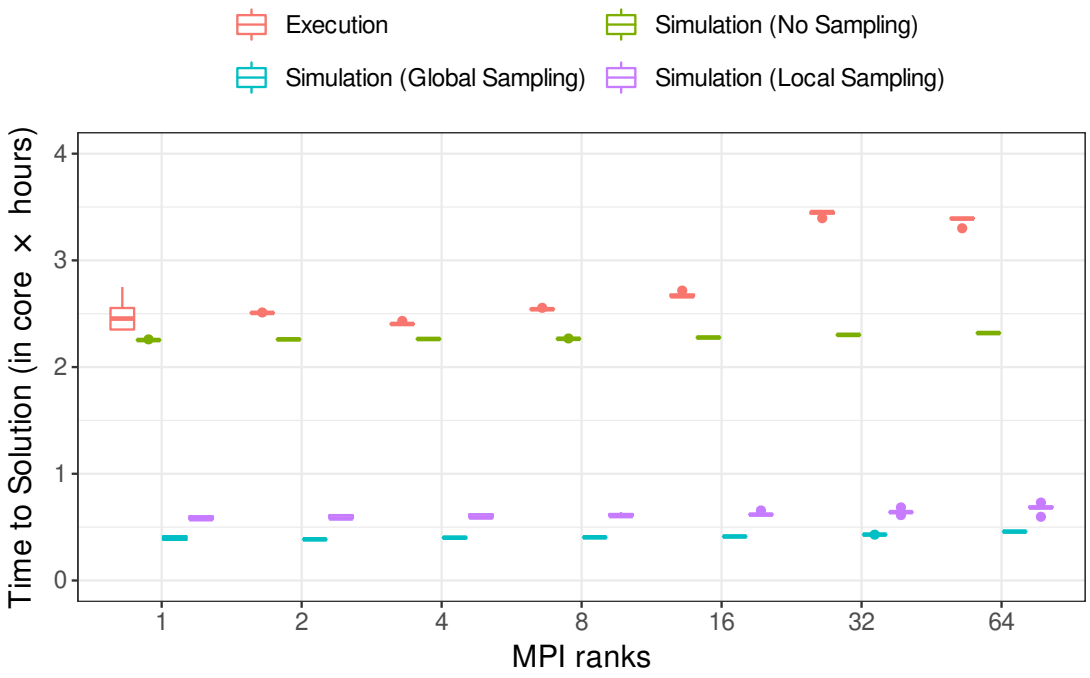


Figure 2: Time to run or simulate (with or without kernel sampling) the execution of a 3D Lennard-Jones melt on a 70^3 region with ExaMiniMD. Each rank runs a single Kokkos thread and is mapped on a different core.

We can see that the number of core × hours needed to solve this problem instance remains stable as the number of MPI ranks increases, i.e., the actual execution completes faster with higher rank counts. The simulation of ExaMiniMD with SMPI runs on a single core and takes the same time to complete whatever the number of MPI ranks used. This simulation time is commensurate to that of the actual execution, but uses much less computing resources. Activating the kernel sampling, either local or global, in the simulation reduces the time to solution by a factor of 5, thus results can be obtained in about 25-30 minutes instead of 2.5 hours.

Accuracy. Figure 3 shows the accuracy of the simulation of an unmodified version of ExaMiniMD with SMPI and the impact of kernel sampling on this accuracy. SMPI is able to correctly reflect the behavior of the simulated application with less variability and a reasonable error. Activating the kernel sampling, be it local or global, slightly degrades the accuracy of the simulation. However, this degradation remains stable as the number of ranks increases. Then, it can easily be taken into account when assessing the performance of a given *in-situ* scheduling strategy with Sim-Situ.

To assess the quality of our simulation on larger core counts, scaling up to the full size of the target platform (i.e., 32 nodes and 1,024 cores), we simulated a larger problem instance with a $90 \times 90 \times 90$ region and $12,000$ iterations. Figure 4 shows that the accuracy of the local sampling version drops from 512 cores while the global sampling version and the simulation without sampling remain accurate.

4.2 Data Analytics Component

Data analytics is the most versatile component of *in-situ* workflows. Users can act on many parameters related to this component when designing and executing their workflows. Depending on what they

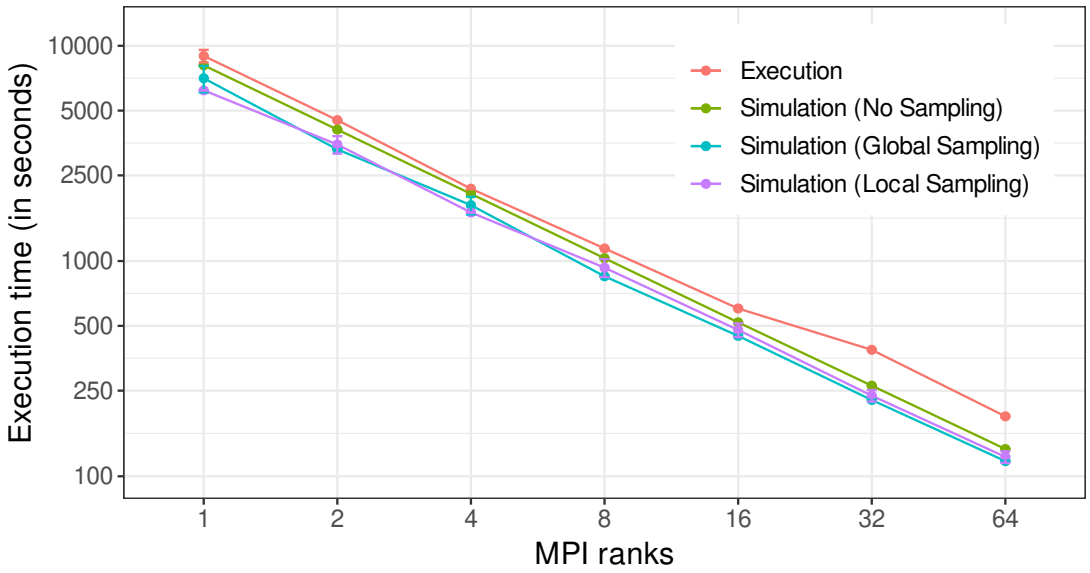


Figure 3: Accuracy of the simulation of 3D Lennard-Jones melt on a 70^3 region (with or without kernel sampling) when varying the number of MPI ranks.

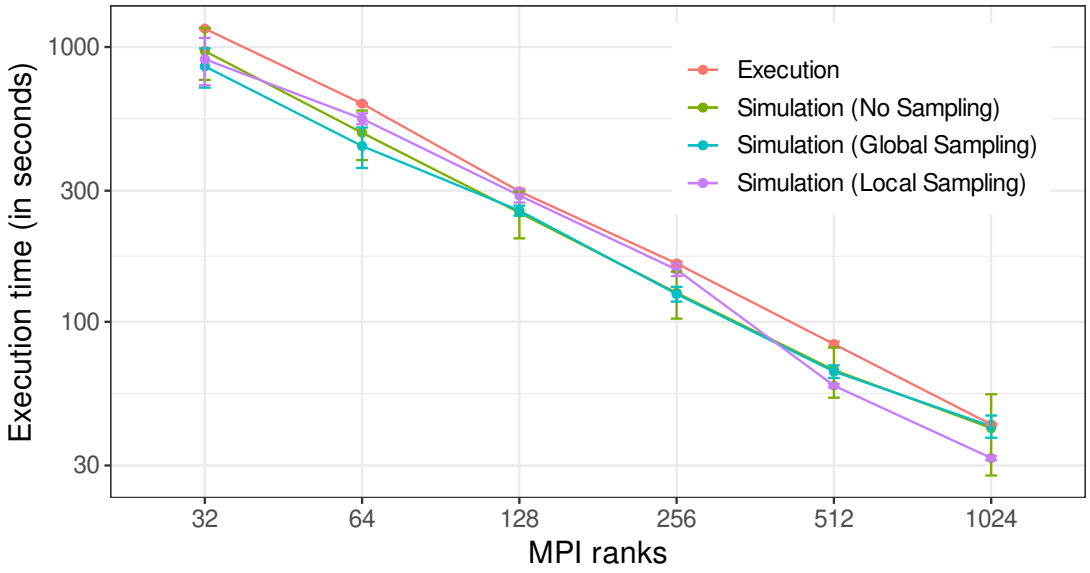


Figure 4: Accuracy of the simulation of 3D Lennard-Jones melt on a 90^3 region (with or without kernel sampling) when varying the number of MPI ranks.

want to observe or steer during the execution of the simulation component, the compute cost and complexity of the data analytics function, the volume of data to transfer from simulation to analytics, and the frequency of the analysis can change from one run to another. Combining these variable parameters leads to interesting questions such as "Is it more efficient to run frequent but light analyses or scarce but heavy ones?". Moreover, opting for a given configuration will have a direct impact on performance and may benefit of a particular allocation and mapping scheme.

Configuration. To reflect this versality of the data analytics component and help users in determining the best allocation and mapping for a given configuration, we decided to augment ExaMiniMD with an extra command line flag (`--analysis`). This allows us to configure the *in-situ* version of ExaMiniMD without having to recompile the application. This flag requires six parameters:

- The number of analytics actors to spawn;
- A file, similar the MPI hostfile, that describes the mapping of the analytics actors;
- The cost per particle of the analytics. The analytics actors then multiply this value by the number of analyzed particles to determine the amount of work they have to simulate;
- A computing scaling factor. This parameter allows us to artificially increase the analytics cost to study different what-if scenarios;
- The size per particle of the data transferred from the simulation component to the analytics component. Similarly, this value is multiplied by the number of analyzed particles;
- A data transfer scaling factor. This parameter allows us to artificially increase the transferred data size to study different what-if scenarios.

In-situ Analytics. We slightly modified the code of ExaMiniMD to plug the proposed simulated analytics component. It is implemented as an external shared library that is linked to ExaMiniMD at compile time and is made of two different types of SimGrid actors. The behavior of the *analytics* actors, whose number can be configured on command line, is described by Algorithm 1. They are in charge of the simulated execution of the analytics function with is computation of the different metrics computed every `thermo` steps.

Algorithm 1 Analytics actor

```
1: loop
2:   Get system state from the DTL
3:   if Poisoned value then
4:     if Last actor running then
5:       Send poisoned value to metric collector
6:     Return
7:   Compute analytics
8:   Send computed metrics to metric collector
```

Each *analytics* actor runs an infinite loop in which it waits for data to analyze to be available in the DTL. When it is the case, the actor simulates the data analytics function. Thanks to the flexible and expressive API of SimGrid, simple functions that simulate the execution of the amount of work given as parameter at the compute speed of the node the actor is mapped on, or more complex data analytics, including complex communication patterns and multi-node allocations, can be simulated. In the latter case, communications of the simulation and analytics components share the same network and contention will be captured by Sim-Situ if some occurs. We simulate the three functions of ExaMiniMD analytics component using the former scenario.

Then the analytics actor asynchronously sends dummy results to the *metric collector* actor, and waits again for new data. At the end of the execution of the simulation component, a poisoned value is sent to all the analytics actors to properly stop them. The last actor running then stops the metric collector by sending it a poisoned value (lines 4-5).

The objective of the *metric collector* (Algorithm 2) is to simulate the accumulation of the metrics done in the analytics phase. As the analytics is decoupled from the MPI simulation in Sim-Situ, a communication scheme, that only implies the SimGrid actors and not the MPI ranks, can be implemented.

Algorithm 2 Metric collector actor

```
1:  loop
2:    n_collected_values = 0
3:    repeat
4:      Get metrics from analytics actors
5:      if Poisoned value then
6:        Return
7:      Accumulate metrics
8:    until n_collected_values = n_ranks
9:    for all n_ranks do
10:     Put a copy of accumulated metrics into the DTL
```

This actor simply waits for having received as many individually computed metrics to accumulate as there are ranks executing the simulation component (lines 3-8). As the number of analytics actors can be smaller than the number of MPI ranks, an actor can send more than one set of metrics to the metric collector. Once all the metric values for a given analytics phase have been collected, this actor puts as many copies of the accumulated values into the DTL (lines 10-11), so that each MPI rank can retrieve one set of metrics and pursue its execution of the simulation component.

4.3 Data Transport Layer

The structure of the proposed Data Transport Layer (DTL) and the data exchanges between the simulation and analytics components of the *in-situ* workflow are illustrated by Figure 5. It is organized around two distinct message queues. The former stores the current system states sent by each of the MPI ranks to the analytics actors (plain arrows) while the latter stores the metrics computed by the analytics actors and aggregated by the metric collector that are sent back to the MPI ranks (dashed arrows). The communications between the analytics and metric collector actors rely on a standard SimGrid mailbox (dotted arrows), and are thus outside the MPI world.

These two message queues are accessed through a Producer-Consumer mechanism provided by SimGrid. This reduces the amount of synchronization needed between the simulation and the analytics components to a minimum. The MPI ranks ingest their current system state into the DTL in a fire-and-forget mode and then immediately proceed with the next iteration of the main MD simulation loop. They will then block to retrieve the analysis results, but only after `thermo` iterations, i.e., before having to start a new analysis. This Producer-Consumer mechanism also improves the flexibility of Sim-Situ as it allows us to start any number of analytics actors without having to further modify the original code of the application.

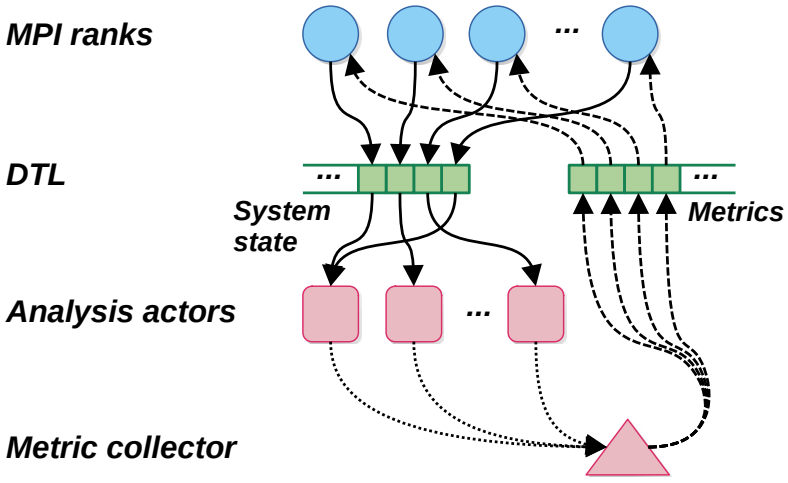


Figure 5: Data exchanges between the Simulation and Analytics components through the Data Transport Layer.

5 Evaluation of In-Situ Allocation and Mapping Strategies with Sim-Situ

In this section, we introduce a model and a metric to assess the performance of different allocation and mapping schemes and illustrate how the Sim-Situ framework can be used[2] to study their impact on the performance of the MD *in-situ* workflow detailed in the previous section.

5.1 Modeling the ExaMiniMD In-Situ Workflow

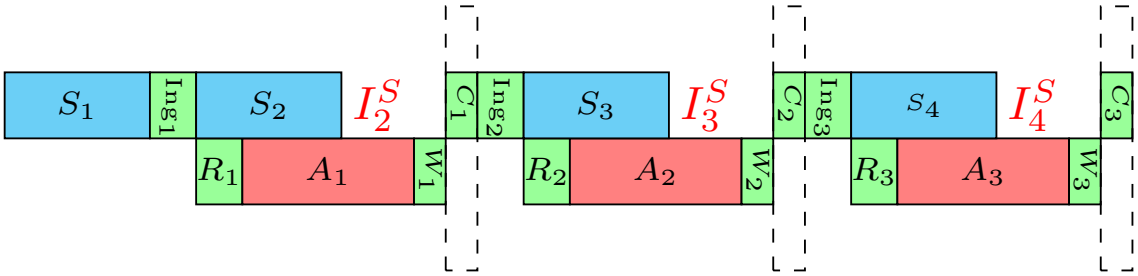
The following execution model is adapted from the one proposed in [10]. The main difference lies in the data transfer pattern between the simulation and analytics components. In the model in [10], the simulation component only sends data to the analytics once, with some buffer constraints imposed by the DTL, while in the considered *in-situ* workflow, the the results of analytics component have to be received before starting a new analysis.

Such a model can help users to obtain a first approximation of the impact of the different configuration parameters of the *in-situ* workflow (i.e., cost and frequency of analysis) and of the allocation and mapping scheme (i.e., number of nodes and core allocation ratio) on the performance of the application. Then, faithful simulations with Sim-Situ can complement this approximation by taking into account more complex phenomena that are captured by the simulator (e.g., contention over the network, load imbalance, etc.).

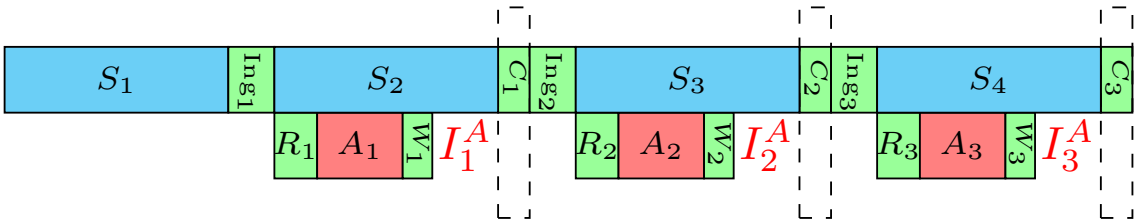
Workflow Stages and Notations. Figure 6 shows the different stages of the execution of the ExaMiniMD *in-situ* workflow. The simulation component (S) produces the data and ingests it into the DTL (Ing). Then, the analytics component (A) retrieves this data from the DTL (R) and processes it. Once done, the analytics component sends the analysis results back to the simulation component (W), which collects (C) them before proceeding with its execution.

This sequence of stages, representing an execution *step*, is repeated iteratively until the completion of the application. Then, we respectively denote as S_i, Ing_i, R_i, A_i, W_i, and C_i, the simulation, ingestion, retrieving, analytics, sending, and collection stages at step i. In this model, S_i actually corresponds to the execution of a certain number of iterations of the main loop of ExaMiniMD between two analyses. As mentioned earlier, this number of iterations is denoted as the *stride* which determines

[2] A reproducibility artifact for this paper is available online.



(a) IS Scenario: $R_i + A_i > S_i + \text{Ing}_i$



(b) IA scenario: $S_i + \text{Ing}_i > R_i + A_i$

Figure 6: Two execution scenarios for the ExaMiniMD *in-situ* workflow.

how frequently the data analytics component is called. Then, for a total number of iterations of the MD simulation loop N and a given stride T, the number of steps in our model will be denoted by $\rho = \frac{N}{T}$, and we thus have $\sum_{i=1}^{\rho} S_i = S$ (the same applies for the other stages). Finally, we make the hypothesis that the different stages are consistent across steps. This means that the time to execute each stage remains constant over all the ρ steps of the application. This hypothesis holds when the number of steps is large enough ($\rho \geq 3$), and the impact of warming-up steps is negligible [10].

Dependencies Between Stages. The data flow within a given step imposes the following precedence constraints among the different stages:

$$S_i \rightarrow \text{Ing}_i \rightarrow R_i \rightarrow A_i \rightarrow W_i \rightarrow C_i, \text{ for } 1 \leq i \leq \rho. \tag{1}$$

Our model also enforces that an analytics stage cannot begin before the simulation component has received the results of the previous analytics phase. This means that, from the second step, the ingestion Ing_i can be done if and only if the collection of the metrics from the previous step C_{i-1} by the simulation component has been done:

$$C_{i-1} \rightarrow \text{Ing}_i, \text{ for } 2 \leq i \leq \rho. \tag{2}$$

Idle Periods. Due to the dependencies between stages described above, idle periods can appear in the execution flow of the application if one of the two main components has to wait for the other. Figure 6 illustrates two execution scenarios, *Idle Analytics* and *Idle Simulation* in which some idle time occurs. The former (IA) corresponds to a case where the execution time of the simulation related stages (i.e.,, simulation and data ingestion) is greater than that of the analytics related stages (i.e., data retrieving and analysis), that is $S_i + \text{Ing}_i > R_i + A_i$. Conversely, the later (IS) corresponds to analytics stages that take more time than the simulation stages, i.e., $R_i + A_i > S_i + \text{Ing}_i$. We denote by I_i^S (resp. I_i^A) the corresponding idle time for the simulation (resp. analytics) component at step i. Then, we reformulate Equation 1 to include these potential idle times:

$$S_i \rightarrow I_i^S \rightarrow \text{Ing}_i \rightarrow R_i \rightarrow A_i \rightarrow W_i \rightarrow I_i^A \rightarrow C_i \tag{3}$$

To simplify the notations, we introduce $S_* = S_i$, for all $i \leq \rho$. A similar simplification is applied to the other stages in Equation 3.

Assuming that the cost of C_* and W_* can be neglected because of the size of the exchanged data, i.e., we consider these stages as synchronization points, and using the constraints expressed in Equation 2 and 3, we define the total idle time I_* of a step as:

$$I_* = I_*^S + I_*^A = \begin{cases} S_i + \text{Ing}_i - (R_i + A_i) & \text{if } I_*^S = 0 \\ R_i + A_i - (S_i + \text{Ing}_i) & \text{if } I_*^A = 0 \end{cases} = |S_i + \text{Ing}_i - (R_i + A_i)| \quad (4)$$

Then, an efficient execution of an *in-situ* workflow does not induce any idle time, hence $I_*^S = I_*^A = 0$. In other words, such an *idle-free* execution is obtained when $S_* + \text{Ing}_* = R_* + A_*$.

In-Situ Workflow Execution Efficiency Metric. Finally, we use the model above to refine a metric presented in [10] to evaluate the performance of *in-situ* workflow executions. To this end, we define the makespan m of the *in-situ* workflow as the sum of the makespan of each step $i \leq \rho$:

$$m = \sum_{i=1}^{\rho} m_i,$$

where m_i is the maximum of the makespan of each component. Then we have:

$$m = \rho \times \max\left(S_* + \text{Ing}_*, R_* + A_*\right) \quad (5)$$

Using this makespan definition, we compute an efficiency ratio η that depends on the total idle time induced during the workflow execution. This ratio is defined as:

$$\eta = 1 - \frac{\rho \times I_*}{m} = 1 - \frac{|S_i + \text{Ing}_i - (R_i + A_i)|}{\max\left(S_* + \text{Ing}_*, R_* + A_*\right)} \quad (6)$$

5.2 Assessing the Efficiency of In Situ Workflow Executions

For this evaluation, we consider a 3D Lennard-Jones melt on a $70 \times 70 \times 70$ region problem instance. The main MD simulation loop has 8,000 iterations. A performance analysis of ExaMiniMD allowed us to set the compute cost per particle to 7.93*e*-7 and the data size per particle to transfer to 100.

For the allocation and mapping strategies, we base our study on the work of Malakar *et al.* [22] in which they set the simulation to analysis core allocation ratio R as the number of cores allocated to the simulation component over the number of cores allocated to the data analytics components. As our target cluster has 32 cores per node, we consider 5 values for this ratio as shown in Table 1. Then, we run simulations for 1, 2, 4, and 8 nodes (i.e., 32, 64, 128, and 256 cores). All the results presented hereafter were obtained by running simulations on a single core of the Dahu cluster.

Table 1: Considered simulation to analysis core allocation ratios.

R	**# simulation cores**	**# analysis cores**
1	16	16
3	24	8
7	28	4
15	30	2
31	31	1

Impact of Simulation to Analysis Core Allocation Ratio. Let's consider a user who would like to perform a constant amount of analysis during the execution of their workflow and know, for a given number of cores, what would be the most efficient simulation to analysis core allocation ratio to use. This user can act on two parameters to execute the desired amount of analysis: the *stride* and the *cost*

of one execution of the data analytics component. For instance, if the MD simulation loop is executed 8,000 times and 400 units of analysis have to be performed, (at least) four (stride, cost) configurations can be envisioned: (20, 1), (200, 10), (500, 25), and (1000, 50). Thanks to the flexibility of Sim-Situ, generating a version of the ExaMiniMD *in-situ* workflow with a larger analysis cost simply amounts to changing one of the command line parameter, i.e., the computing scaling factor.

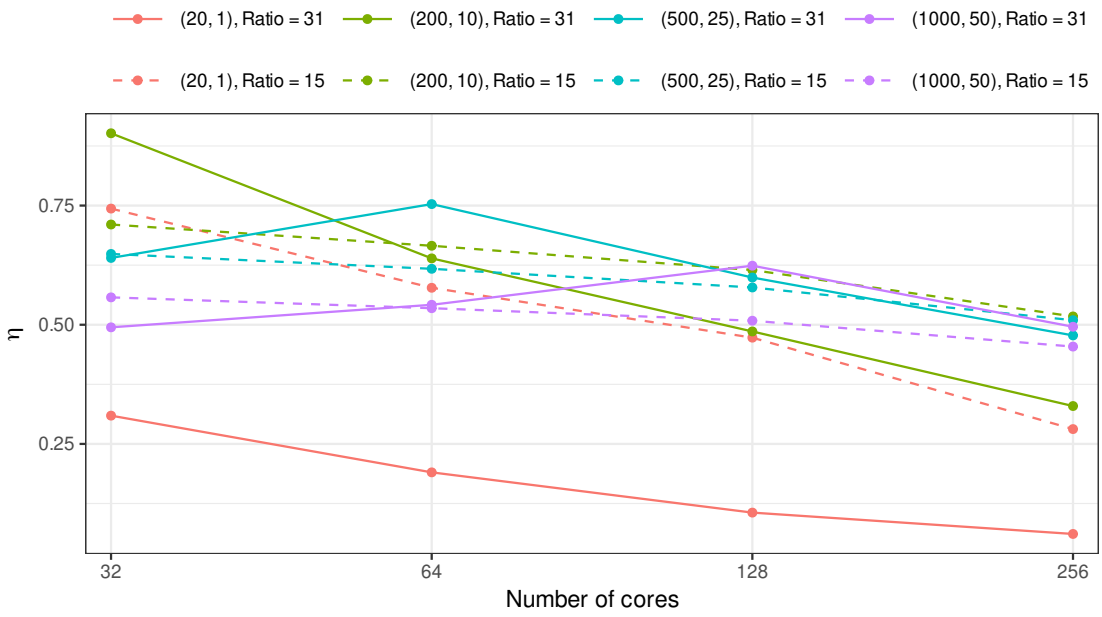


Figure 7: Efficiency of ExaMiniMD *in-situ* workflow in four (stride, analytics cost) configurations for two the core allocation ratios.

Figure 7 shows the achieved efficiency for these four combinations of stride and analysis costs and two core-allocation ratios ($R = 15$ and $R = 31$). The other ratios lead to lower efficiency for any core count and are thus not displayed for the sake of readability. It shows some interesting trends and tradeoffs. We can see that the (stride, cost) configuration that leads to the best efficiency is not the same for every core count. As the number of cores available for the execution of the *in-situ* workflow increases, it appears to be more efficient to reduce the frequency and increase the cost of the analytics component. This is confirmed by the trends of the (20, 1) and (200, 10) configurations with $R = 31$ whose efficiency steadily decreases with the increase of the number of cores. For these configurations, the analytics actors do not have enough work to process and thus are idle most of the time. As the total core count grows, more analytics actors are started, hence amplifying this phenomenon. A similar trend can be seen for larger analytics cost, but the tipping point where the efficiency starts to drop is for larger core counts.

We also observe a generally decreasing trend for the efficiency as the number of cores grows with $R = 15$, but over a narrower range. Moreover, the (200, 10) configuration appears to be consistently achieving good efficiency for this core allocation ratio, for all total core counts. This better stability might be preferred by users when they have to adapt their executions to the number of currently available cores they have access to and do want to risk to loose efficiency by selecting the wrong configuration.

Figure 8 shows a different view of the (1000, 50) configuration, i.e., the evolution of the active and idle times of the simulation and analytics components when the core allocation ratio and the total

number of cores increase[3].

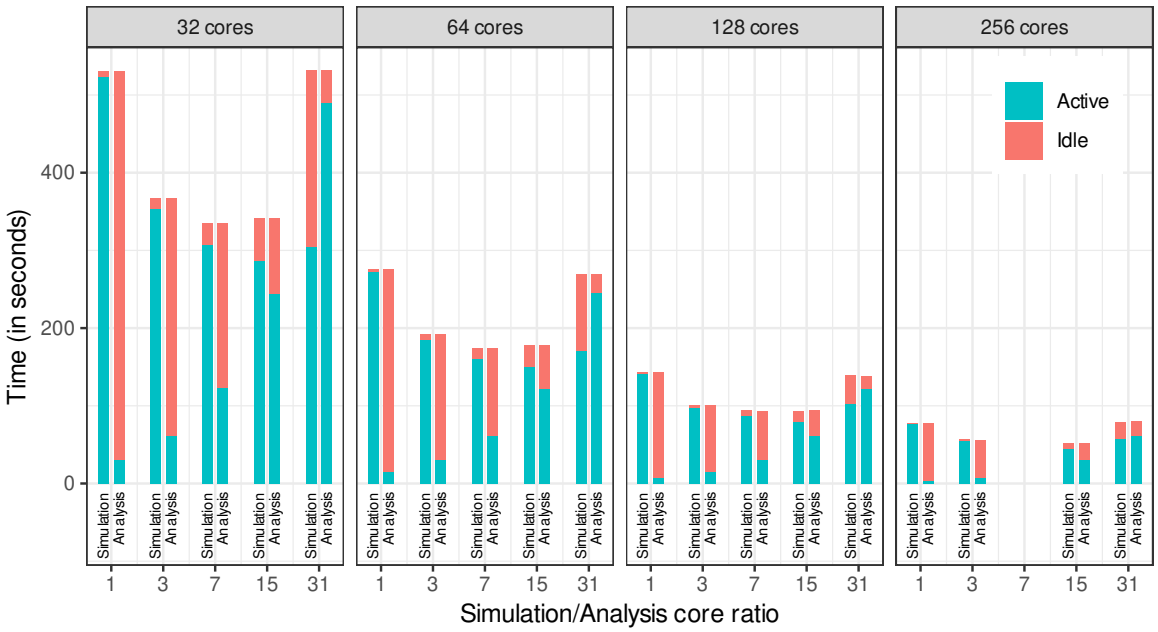


Figure 8: Evolution of the active and idle times of the simulation and analytics components when increasing the core allocation ratio and the total number of cores for the (1000, 50) scenario.

In this scenario, we can see that the respective active times of the simulation and analytics component follow opposite trends. For small core allocation ratios, the execution time is completely dominated by the time to execute the MD simulation. Then, the time to execute the analytics component increases linearly with the ratio, as less cores are allocated to execute the same amount of analysis, until a tipping point is reached where the simulation waits for analytics ($R = 31$).

We also see that some "sweet spots" can be found, typically for $R = 15$, where the active times of the simulation and analytics components are both efficient and well balanced. It is interesting to note that it is in contradiction with the efficiency metric that indicates a better efficiency with a core allocation ratio of 31 from 64 cores.

These results illustrate how Sim-Situ can be used to determine a good core-allocation ratio for a given cost of analysis and number of nodes and that it is important for users of the Sim-Situ framework to leverage all the metrics the tool can provide them while designing their workflow.

Impact of simulation to analysis data transfers. Another design choice faced by users of *in-situ* workflows is to decide of the mapping of the resources allocated to the data analytics component. In other words, the question is: "would it better to adopt an *in-situ* strategy, i.e., mapping the analytics resources on the same nodes as the simulation resources, or an *in-transit* strategy, i.e., dedicating some node(s) to the analytics?". The former has the advantage of minimizing the cost of data exchanges between simulation and analytics thanks to a shared memory space, but the scattering of the analytics resources over multiple nodes may hinder the performance of this component (e.g., communication intensive analytics function). Conversely, the latter benefits of having all the analytics located on a single, or small number of dedicated nodes, but induces a larger communication overhead to exchange data with the simulation component.

[3]The lack of values for 256 cores and $R = 7$ is due to a crash of ExaMiniMD for this particular instance with 224 MPI ranks (w/ or w/o Sim-Situ). It seems to come from a badly handled division by 0 in ExaMiniMD's code.

To illustrate how Sim-Situ can help users to evaluate the relative performance of *in-situ* and *in-transit* mapping schemes, we consider the following scenario. The simulation component still corresponds to the main loop of ExaMiniMD. The analytics component now corresponds to a function that involves all the analytics resources and whose performance is impacted by the number of nodes onto which these resources are allocated, i.e., its execution time increases with the resource scattering. Finally, the user can decide of the volume of data produced by the simulation to transfer to the analytics.

Simulating such a performance study is made easy by the features of Sim-Situ. Switching from *in-situ* to *in-transit* mappings simply amounts to change the analytics hostfile, while changing the volume of transferred data or the performance profile of the analytics function can be done on command line. Figure 9 shows the evolution of the execution time of the simulation component of the workflow when the volume of data to exchange with the analytics component is scaled up to a thousand times. The workflow is executed on 16 nodes and two execution modes are considered. The *in-situ* mode uses a core allocation ratio of 15, i.e., two cores per node are allocated to the analytics component while a full node is dedicated to the analysis in the *in-transit* mode.

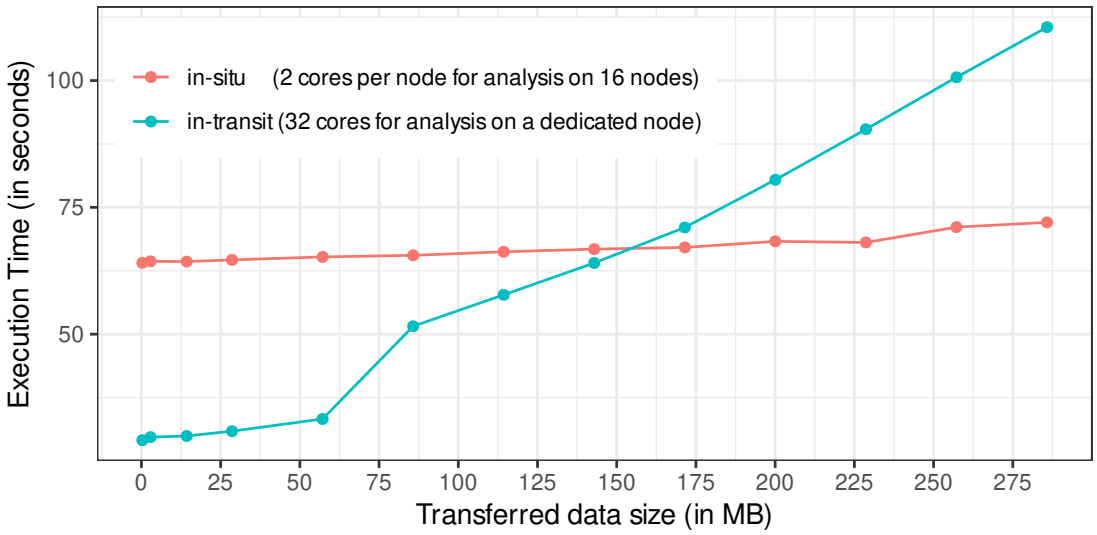


Figure 9: Evolution of the execution time of the simulation component when the volume of data transfer is scaled up in the *in-situ* and *in-transit* execution modes with 16 nodes and $R = 15$.

We can see that the scattering of the analytics resources across nodes makes the *in-situ* execution mode less efficient than the *in-transit* execution mode when a small amount of data is exchanged. However, as we increase this volume of transferred data, the execution time of the *in-transit* mode starts to increase linearly, while the *in-situ* execution mode is only slightly impacted as the data exchanges are done through a shared memory space. Such simulation-based studies may help users in the design of their analytics component by showing them where lies the tipping point between *in-transit* and *in-situ* modes for a given configuration of their workflow.

6 Conclusion and Future Work

Analyzing the results of large-scale numerical simulations as they are produced is an appealing alternative to classical *post-hoc* analyses that are more and more impacted by the increasing discrepancy between the relative performance of computing and storage subsystems in extreme-scale supercomputers. However, the development of such *in-situ* scientific workflows raises several challenging questions,

such as "what *amount* of analysis can be done and at which *frequency*?", "how many *resources* can be taken off of the execution of the simulation to execute the analysis?", or "Is it better to perform *in-situ* or *in-transit* analytics?".

Determining answers to these questions that do not cause the performance of an *in-situ* workflow to be worse than the classical "simulation then analysis" approach usually falls down to evaluating the performance of different allocation and mapping strategies for different input configurations. However, the state-of-the-art on the performance evaluation of *in-situ* workflows shows that it relies either on time- and resource-consuming experiments on a limited set of scenarios or on the simulation of abstracted versions of the initial applications that may lack of realism.

In this paper, we introduced the Sim-Situ framework, a generic simulation framework for *in-situ* workflows based on the popular SimGrid toolkit. The modular design of Sim-Situ faithfully captures the structure of classical *in-situ* workflows and the behavior of their different components. We illustrated its simulation capacities on the ExaMiniMD Molecular Dynamics proxy-application. With only a few minor code modifications, we showed how Sim-Situ could be used to study different execution scenarios of *in-situ* workflows and highlight important performance tradeoffs.

As part of our future work, we plan to further demonstrate the simulation capacities of Sim-Situ by investigation more allocation and mapping strategies on different use-case applications. We will particularly focus on *in-transit* processing where nodes are dedicated to analytics. Studying such strategies would be a first step in evaluating the impact of data transfers and network performance on the execution of scientific workflows. We also plan to extend the simulation capacities and realism of Sim-Situ by developing more complex versions of the Data Transport Layer component that mimic the behavior of popular implementations such as DataSpaces [14] or Dimes [15]. The objective is to provide Sim-Situ users with the capacity to select which flavor of the DTL they want to use in the simulation of their *in-situ* workflows. Finally, we plan to leverage Sim-Situ to carry out performance evaluations in scenarios that would be hardly possible to evaluate through actual experiments on supercomputers. For instance, the necessity of running series of parallel MD simulations in ensembles [23] broadens the range of feasible *in-situ* configurations and raises new allocation and scheduling challenges. Moreover, the modularity of the Sim-Situ framework offers enough flexibility to envision the online evaluation of scheduling decisions in the context of an adaptive sampling process [24].

Acknowledgments. Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations (see https://www.grid5000.fr). This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.

References

[1] R. Gerber, J. Hack, K. Riley, K. Antypas, R. Coffey, E. Dart, T. Straatsma, J. Wells, D. Bard, S. Dosanjh, I. Monga, M. E. Papka, and L. Rotman, "Crosscut report: Exascale Requirements Reviews, March 9–10, 2017 – Tysons Corner, Virginia. An Office of Science review sponsored by: Advanced Scientific Computing Research, Basic Energy Sciences, Biological and Environmental Research, Fusion Energy Sciences, High Energy Physics, Nuclear Physics," 2018. [Online]. Available: https://www.osti.gov/biblio/1417653

[2] H. Casanova, A. Giersch, A. Legrand, M. Quinson, and F. Suter, "Versatile, Scalable, and Accurate Simulation of Distributed Applications and Platforms," *Journal of Parallel and Distributed Computing*, vol. 74, no. 10, pp. 2899 – 2917, 2014.

[3] F. Zheng, H. Zou, G. Eisenhauer, K. Schwan, M. Wolf, J. Dayal, T.-A. Nguyen, J. Cao, H. Abbasi, S. Klasky, N. Podhorszki, and H. Yu, "FlexIO: I/O Middleware for Location-Flexible Scientific Data Analytics," in *Proc. of the 27th IEEE International Symposium on Parallel and Distributed Processing*, Boston, MA, 2013, pp. 320–331.

[4] P. Malakar, V. Vishwanath, T. Munson, C. Knight, M. Hereld, S. Leyffer, and M. E. Papka, "Optimal Scheduling of In-Situ Analysis for Large-Scale Scientific Simulations," in *Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Austin, TX, Nov. 2015.

[5] P. Malakar, V. Vishwanath, C. Knight, T. Munson, and M. E. Papka, "Optimal Execution of Co-analysis for Large-Scale Molecular Dynamics Simulations," in *Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Salt Lake City, UT, Nov. 2016, pp. 702–715.

[6] Q. Sun, T. Jin, M. Romanus, H. Bui, F. Zhang, H. Yu, H. Kolla, S. Klasky, J. Chen, and M. Parashar, "Adaptive Data Placement for Staging-Based Coupled Scientific Workflows," in *Proc. of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Austin, TX, Nov. 2015, pp. 1–12.

[7] P. Subedi, P. E. Davis, and M. Parashar, "Leveraging Machine Learning for Anticipatory Data Delivery in Extreme Scale In-situ Workflows," in *Proc. of the IEEE International Conference on Cluster Computing*, Albuquerque, NM, Sep. 2019, pp. 1–11.

[8] E. Lohrmann, Z. Lukić, D. Morozov, and J. Müller, "Programmable In Situ System for Iterative Workflows," in *Proc. of the 21st Workshop on Job Scheduling Strategies for Parallel Processing*. Orlando, FL: Springer, 2018, pp. 122–131.

[9] G. Aupy, B. Goglin, V. Honoré, and B. Raffin, "Modeling High-Throughput Applications for In Situ Analytics," *International Journal of High Performance Computing Applications*, vol. 33, no. 6, pp. 1185–1200, 2019.

[10] T. M. A. Do, L. Pottier, S. Caíno-Lores, R. Ferreira da Silva, M. A. Cuendet, H. Weinstein, T. Estrada, M. Taufer, and E. Deelman, "A Lightweight Method for Evaluating In Situ Workflow Efficiency," *Journal of Computational Science*, vol. 48, p. 101259, 2021.

[11] P. Velho, L. M. Schnorr, H. Casanova, and A. Legrand, "On the Validity of Flow-Level Tcp Network Models for Grid and Cloud Simulations," *ACM TOMACS*, vol. 23, no. 4, Dec. 2013.

[12] A. Degomme, A. Legrand, G. Markomanolis, M. Quinson, M. Stillwell, and F. Suter, "Simulating MPI applications: the SMPI approach," *IEEE Transactions on Parallel and Distributed Systems*, vol. 18, no. 8, pp. 2387–2400, 2017.

[13] P. Bédaride, A. Degomme, S. Genaud, A. Legrand, G. Markomanolis, M. Quinson, M. Stillwell, F. Suter, and B. Videau, "Toward Better Simulation of MPI Applications on Ethernet/TCP Networks," in *Proc. of the 4th International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, Denver, CO, Nov. 2013.

[14] C. Docan, M. Parashar, and S. Klasky, "DataSpaces: an Interaction and Coordination Framework for Coupled Simulation Workflows," *Cluster Computing*, vol. 15, no. 2, pp. 163–181, 2012.

[15] F. Zhang, T. Jin, Q. Sun, M. Romanus, H. Bui, S. Klasky, and M. Parashar, "In-memory Staging and Data-Centric Task Placement for Coupled Scientific Simulation Workflows," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 12, p. e4147, 2017.

[16] A. Thompson and C. Trott, "A Brief Description of the Kokkos implementation of the SNAP potential in ExaMiniMD," Office of Scientific and Technical Information, Tech. Rep. 1409290, Nov. 2017.

[17] "ExaMiniMD Proxy Application GitHub Repository," [Online] https://github.com/ECP-copa/ExaMiniMD, May 2021.

[18] O. Aaziz, C. Vaughan, J. Cook, J. Cook, J. Kuehn, and D. Richards, "Fine-Grained Analysis of Communication Similarity between Real and Proxy Applications," in *Proc. of the 10th IEEE International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems*, Denver, CO, Nov. 2019.

[19] S. Plimpton, "Fast Parallel Algorithms for Short-Range Molecular Dynamics," *Journal of Computational Physics*, vol. 117, no. 1, pp. 1–19, 1995.

[20] T. Cornebize, "High Performance Computing: towards better Performance Predictions and Experiments," Ph.D. dissertation, Université Grenoble-Alpes, Grenoble, France, Jun. 2021.

[21] D. Richards, O. Aaziz, J. Cook, H. Finkel, B. Homerding, T. Juedeman, T. McCorquodale, Peter an Mintz, and S. Moore, "Quantitative Performance Assesment of Proxy Apps and Parents," Lawrence Livermore National Laboratory, Tech. Rep. LLNL-TR-750182, Apr. 2018.

[22] P. Malakar, T. Munson, C. Knight, V. Vishwanath, and M. E. Papka, "Topology-Aware Space-Shared Co-Analysis of Large-Scale Molecular Dynamics Simulations," in *Proc. of the International Conference for High Performance Computing, Networking, Storage, and Analysis*, Dallas, TX, Nov. 2018.

[23] R. Chelli and G. F. Signorini, "Serial generalized ensemble simulations of biomolecules with self-consistent determination of weights," *Journal of Chemical Theory and Computation*, vol. 8, no. 3, 2012.

[24] E. Hruska, V. Balasubramanian, H. Lee, S. Jha, and C. Clementi, "Extensible and Scalable Adaptive Sampling on Supercomputers," *Journal of Chemical Theory and Computation*, vol. 16, no. 12, pp. 7915–7925, 2020.