Journal of Computational Science 48 (2021) 101259

Contents lists available at ScienceDirect

Journal of Computational Science

journal homepage: www.elsevier.com/locate/jocs

A lightweight method for evaluating *in situ* workflow efficiency

Tu Mai Anh Do[a,*], Loïc Pottier[a], Silvina Caíno-Lores[b], Rafael Ferreira da Silva[a], Michel A. Cuendet[c], Harel Weinstein[c], Trilce Estrada[d], Michela Taufer[b], Ewa Deelman[a]

[a] *Information Sciences Institute, University of Southern California, Marina del Rey, CA, USA*
[b] *University of Tennessee at Knoxville, Knoxville, TN, USA*
[c] *Weill Cornell Medicine, Cornell University, New York, NY, USA*
[d] *University of New Mexico, Albuquerque, NM, USA*

ARTICLE INFO

Keywords:
Scientific workflow
In situ model
Molecular dynamics
High-performance computing

ABSTRACT

Performance evaluation is crucial to understanding the behavior of scientific workflows. In this study, we target an emerging type of workflow, called *in situ* workflows. These workflows tightly couple components such as simulation and analysis to improve overall workflow performance. To understand the tradeoffs of various configurable parameters for coupling these heterogeneous tasks, namely simulation stride, and component placement, separately monitoring each component is insufficient to gain insights into the entire workflow behavior. Through an analysis of the state-of-the-art research, we propose a lightweight metric, derived from a defined *in situ* step, for assessing resource usage efficiency of an *in situ* workflow execution. By applying this metric to a synthetic workflow, which is parameterized to emulate behaviors of a molecular dynamics simulation, we explore two possible scenarios (Idle Simulation and Idle Analyzer) for the characterization of *in situ* workflow execution. In addition to preliminary results from a recently published study [11], we further exploit the proposed metric to evaluate a practical *in situ* workflow with a real molecular dynamics application, i.e., GROMACS. Experimental results show that the *in transit* placement (analytics on dedicated nodes) sustains a higher frequency for performing *in situ* analysis compared to the *helper-core* configuration (analytics co-allocated with simulation).

1. Introduction

High performance computing (HPC) is mainstream for enabling the execution of scientific workflows which are composed of complex executions of computational tasks and the constantly growing data movements between those tasks [30]. Traditionally, a workflow describes multiple computational tasks and represents data and control flow dependencies. Moreover, the data produced by the scientific simulation are stored in persistent storage and visualizations or analytics are performed post hoc. This approach is not scalable, mainly due to the fact that scientific workflows are becoming increasingly compute- and data-intensive at extreme-scale [28]. Storage bandwidth has also failed to keep pace with the rapid computational growth of modern processors due to the stagnancy of I/O advancements [33,15]. This asymmetry in I/O and computing technologies, which is being observed in contemporary and emerging computing platforms, prevents post hoc processing from handling large volumes of data generated by large-scale simulations [4]. Therefore, storing the entire output of scientific simulations on disk causes major bottlenecks in workflow performance. From a hardware perspective, moving data consumes more energy than performing the computing operation on the same amount of data [33]. Although computational capacity keeps increasing along with chip technologies, this growth only exacerbates the imbalance between the I/O and the computation portions in applications. To reduce this disparity, several new high-performance memory systems that reside closer to the computation units have been developed (e.g., burst buffers [29], high-bandwidth memory [34], non-volatile memory [14], etc.). These systems create opportunities to overcome the expensive cost of I/O thanks to the low-latency access capability of those advanced storage technologies. Simultaneously, scientists have moved towards a new paradigm for scientific simulations called *in situ*, in which data is visualized and/or analyzed as it is generated [4]. This accelerates simulation

* Corresponding author at: USC Information Sciences Institute, 4676 Admiralty Way Suite 1001, Marina del Rey, CA 90292, USA.
E-mail addresses: tudo@isi.edu (T.M.A. Do), lpottier@isi.edu (L. Pottier), scainolo@utk.edu (S. Caíno-Lores), rafsilva@isi.edu (R. Ferreira da Silva), mac2109@med.cornell.edu (M.A. Cuendet), haw2002@med.cornell.edu (H. Weinstein), estrada@cs.unm.edu (T. Estrada), mtaufer@utk.edu (M. Taufer), deelman@isi.edu (E. Deelman).

https://doi.org/10.1016/j.jocs.2020.101259
Received 16 October 2020; Accepted 21 November 2020
Available online 3 December 2020

I/O by bypassing the file system, pipelining the analysis, and improving the overall workflow performance [9].

An *in situ* workflow describes a scientific workflow with multiple components (simulations with different parameters, visualization, analytics, etc.) running concurrently [33,9], potentially coordinating their executions using the same allocated resources to minimize the cost of data movement [4]. Data periodically produced by the main simulation are processed, analyzed, and visualized at runtime rather than post-processed on dedicated nodes. This approach offers many advantages for processing large volumes of simulated data and efficiently utilizing computing resources. By co-locating the simulation and the analysis kernels, *in situ* solutions reduce the global I/O pressure and the data footprint in the system [15]. To fully benefit from these solutions, the simulation and the analysis components have to be effectively managed so that they do not slow each other down. Therefore, in this paper we study and characterize two specific categories of *in situ* workflows, namely *helper-core* and *in transit* workflows [4]. Specifically, in the *helper-core* workflow, the analysis component is placed on the same node as the simulation; while in the *in transit* workflow, simulation data is staged to a dedicated node where the analysis is allocated. Workflows are required to capture the individual behavior of multiple coupled workflow components (i.e., concurrent executions of overlapped steps with inter-component data dependency). Through the use of a theoretical framework, this paper aims to provide guidelines for the evaluation and characterization of *in situ* workflows. We target a widely-used class of workflows, namely large-scale molecular dynamics (MD) simulations. We argue that the proposed solutions and the lessons learned from the proposed synthetic *in situ* workflows [11] can be directly translated into production *in situ* workflows. We further deploy a practical workflow using a medium-scale all-atom MD simulation run with a popular MD engine to study the impact of the aforementioned component placements on the *in situ* coupling in a realistic setting. To this end, the insights gained from the practical workflow are beneficial for scaling up to *ensemble workflow*, which is a set of different simulation-analysis workflows exploring a combination of scenarios. Our contribution is fivefold:

1. We discuss practical challenges in evaluating next-generation workflows, such as *in situ*. Most of current HPC monitoring tools are not specifically designed for these workflows. Thus, utilizing these tools to evaluate *in situ* workflows is not straightforward. In this work, we generalize *in situ* systems to bring to light the unique characteristics of *in situ* workflows from an evaluation standpoint.
2. We define a non-exhaustive list of imperative metrics that need to be monitored for aiding the characterization of *in situ* workflows. By reviewing the state-of-the-art in profiling tools that are able to collect the metrics, we confirm the feasibility of leveraging this set of metrics in the *in situ* workflow context. Additionally, we highlight our novel approach of measuring idle time for quantifying the efficiency of an *in situ* run.
3. We propose a framework to formalize *in situ* workflow executions based on their iterative patterns. Under the framework's constraints, we develop a lightweight approach that is beneficial when comparing the performance of configuration variations in an *in situ* system. The aim is to provide a lightweight approach that is able to run concurrently with the *in situ* workflow and possibly enable its adaptation at runtime.
4. We provide insights into the behaviors of *in situ* workflows by applying the proposed metric to characterize an MD synthetic workflow. Leveraging the use of limited computing resources for emulating simulation behavior at large scales, we use this synthetic workflow to prove the precision of the proposed metric and identify the *in situ* behavior in a wide range of simulation systems.
5. We further examine the significance of *in situ* placements on coupling performance through employing the proposed metric in a practical workflow using a real high-performance MD application. We claim that our findings are valuable for designing an ensemble of workflows, where multiple independent *in situ* couplings are running at the same time.

2. Background and related work

In situ workflows monitoring. Many monitoring and performance profiling tools for HPC applications have been developed over the past decade such as TAU [27], CrayPat [10], or HPCTOOLKIT [1]. With the advent of *in situ* workflows [5], new monitoring and profiling approaches targeting tightly-coupled workflows have been studied. LDMS [2] is a loosely-integrated scalable monitoring infrastructure that targets general large-scale applications and delivers a low-overhead distributed solution, in contrast to TAU [27], which provides a deeper understanding of the application at a higher computational cost. SOS [36] provides a distributed monitoring platform, conceptually similar to LDMS but specifically designed for online *in situ* characterization of HPC applications. An SOS daemon running on each compute node intercepts events and registers them into a database; the monitored application may fetch data from that database to get feedback. TAU, in association with ML techniques, has been used to tune the parameters of *in situ* simulations and optimize the execution at runtime [39]. ADIOS [20], the next-generation IO-stack, is built on top of many *in situ* data transport layers, e.g., DataSpaces [12] or DIMES [38]. Savannah [15], a workflow orchestrator, has been leveraged to bundle a coupled simulation with two main simulations, multiple analysis kernels, and a visualization service [9]. The performance and monitoring service was provided via SOS and the I/O middleware via ADIOS. These works focus mainly on providing monitoring schemes for *in situ* workflows. Here we propose, instead, a novel method to extract useful knowledge from the captured performance data.

In situ data management. FlexAnalytics [40] optimizes the performance of coupling simulations with *in situ* analytics by evaluating data compression and query over different I/O paths: memory-to-memory and memory-to-storage. A large-scale data staging implementation [37] over MPI-IO operations describes a way to couple with *in situ* analysis using a non-intrusive approach. The analytics accesses data staged to the local persistent storage of compute nodes to enhance data locality. Decaf [13] provides a message-driven dataflow middleware that supports both tight and loose coupling of *in situ* tasks. Our work mainly focuses on in-memory staging and comprehensively characterizes memory-to-memory transfer using RDMA for both within a compute node (*helper-core*) and across nodes (*in transit*). Smart [35] provides *in situ* MapReduce-like interfaces for scientific analytics—a breakthrough in being able to access in-memory simulated data, though an intrusive approach. Our use case prototype has a non-intrusive approach to employ *in situ* analytics through two abstract components called the *ingester* and the *retriever*, described in the next section.

3. General *in situ* workflow architecture

In this section, we describe the architecture of an *in situ* workflow that underlies our study. We also define and motivate a set of non-exhaustive metrics that need to be captured for *in situ* workflow performance characterization.

In situ architecture. In this work, we propose an *in situ* architecture that enables a variety of *in situ* placements to characterize the behavior of *in situ* couplings. Although we focus on a particular type of *in situ* workflows (composed of simulation and data analytics), our approach is broader and applicable to a variety of *in situ* components. For example, *in situ* components could consist of an ensemble of independent simulations coupled together. The *in situ* workflow architecture (Fig. 1) features three main components:

- A *simulation* component that performs computations and periodically generates a snapshot of scientific data.

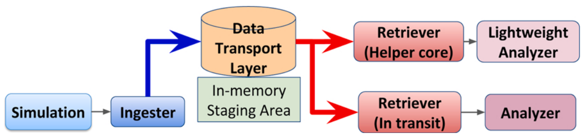


Fig. 1. A general *in situ* workflow software architecture.

- A *data transport layer (DTL)* that is responsible for efficient data transfer.
- An *analyzer* component that applies several analysis kernels to the data received periodically from the simulation component via the DTL.

In general, the data transport layer (DTL) may be implemented using different technologies to enable data delivery between workflow components, e.g., the use of Burst Buffers for *in transit* processing [29], or complex memory hierarchies for *in situ* processing of small volumes of data and a large number of computing jobs [17]. In this paper, we leverage DataSpaces [12] as DTL implementation to enable efficient and scalable in memory data staging among coupled components, but our approach is agnostic from the DTL used.

On the data path "simulation-to-analyzer" depicted on Fig. 1: (1) the *ingester* ingests data from a certain data source and stores them in the DTL and the (2) *retriever*, in a reverse way, gets data from the DTL to perform further operations. These two entry points allow us to abstract and detach complex I/O management from the application code. This approach enables more control in terms of *in situ* coupling and is less intrusive than many current approaches. The ingester synchronously inputs data from the simulation by sequentially taking turns with the simulation using the same resource. The ingester is useful to attach simple tasks to pre-process data (e.g., data reduction, data compression). The architecture allows *in situ* execution with various placements of the retriever. A *helper-core* retriever co-locates with the ingester on a subset of cores where the simulation is running—it asynchronously gets data from the DTL to perform an analysis. As the retriever is using the *helper-core* placement, the analysis should be lightweight to prevent simulation slowdown. An *in transit* retriever runs on dedicated resources (e.g., staging I/O nodes [40]), receives data from the DTL and performs compute-intensive analysis tasks. We compare *helper-core* and *in transit* retrievers in detail in Section 6.

In situ workflow metrics. To characterize *in situ* workflows, we have defined a foundational set of metrics (Table 1). As a first metric, it is natural to consider the makespan, which is defined as three metrics corresponding to time spent in each component: the simulation, the analyzer, and the DTL. The periodic pattern enacted by *in situ* workflows may impose data dependencies between steps of coupled components, e.g., the analyzer may have to wait for data sent by the simulation to become available in the DTL for reading. Thus, we monitor the idle time of each individual component.

Most current HPC monitoring tools, such as TAU [27], HPCToolkit [1], or CrayPat [10] aim to capture the performance profile of standalone applications and, thus their design is inadequate for *in situ* workflows. Table 2 provides an overview of whether these tools can be used to capture the different metrics defined in Table 1. This literature search further underlines the novelty of our utilization of the idle time during the execution to characterize *in situ* workflows because, to the best of our knowledge, no study used this approach. Moreover, in this work, we use TAU for monitoring purposes, mainly due to its versatility, as demonstrated by Table 2.

Table 1
Selected metrics for *in situ* workflows characterization.

Name	Definition	Unit
MAKESPAN	Total workflow execution time	*s*
TIMESIMULATION	Total time spent in the simulation	*s*
TIMEANALYTICS	Total time spent in the analysis	*s*
TIMEDTL	Total time spent in data transfers	*s*
TIMESIMULATIONIDLE	Idle time during simulation	*s*
TIMEANALYTICSIDLE	Idle time during analysis	*s*

Table 2
State-of-the-art tools used for profiling *in situ* applications.

	TAU	HPCToolkit	CrayPat	WOWMON	SOS
MAKESPAN	[16]	[26]	[21]	[39]	
TIMESIMULATION	[39,16,21]	[26]	[21]	[39]	
TIMEANALYTICS	[39,16,21]	[26]	[21]	[39]	
TIMEDTL	[39,16,25]	[26]		[39]	[25]

Recent works (e.g., LDMS [2], SOS [36], and WOWMON [39]) have proposed general-purpose distributed approaches to provide global workflow performance information by aggregating data from each component. Thanks to those advanced frameworks, extracting meaningful profiling data from *in situ* workflows is efficient. However, *in situ* evaluation is still challenging due to the lack of thorough guidelines for using these data to extract meaningful insights on *in situ* workflows. Only a few studies [9,18] have addressed this challenging problem. In this work, we focus on how to use profiling data to characterize the *in situ* workflows. The motivation behind this study is to address the lack of characterization studies for *in situ* workflows.

4. *In situ* execution model

In this work, we propose a novel method to estimate and characterize *in situ* workflows behaviors from collected performance data. To this end, we develop a theoretical framework of *in situ* executions. In this study, we consider a dedicated failure-free platform without any interferences (caches, I/O, network, etc.) from other applications.

4.1. Framework

In traditional workflows, the simulation and the post-processing analyzer are typical components, in which the post-processing follows the simulation in a sequential manner. Let a *stage* be a part of a given component. We can identify two main stages for each component (see Fig. 2):

- Simulation component: (S) is the computational step that produces the scientific data and (W) is the I/O (or DTL) stage that writes the produced data.
- Analytics component: (R) is the DTL stage that reads the data previously written and (A) is the analysis stage.

However, *in situ* workflows exhibit a periodic behavior: S, W, R, and A follow the same sequential order but, instead of operating on all the data, they operate only on a subset of it iteratively. Here, this subset of data is called a *frame* and can be seen as a snapshot of the simulation at a given time t. Let S_i, W_i, R_i, and A_i be respectively, the simulation, the write, the read and the analysis stage at step i, respectively. In other words, S_i produces the frame i, W_i writes the produced frame i into a buffer, R_i reads the frame i, and A_i analyzes the frame i. Note that, an actual simulation computes for a given number of steps, but only a subset of these steps are outputted as frames and analyzed [31]. The frequency of simulation steps to be analyzed is defined by the *stride*. Let n be the total number of simulation steps, s the stride, and m the number of steps actually analyzed and outputted as frames. We have $m = \frac{n}{s}$.

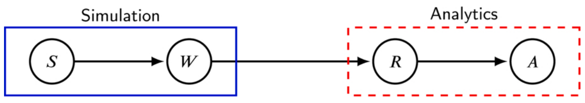


Fig. 2. Classic workflow design featuring two components and four stages.

However, we model the *in situ* workflow itself and not only the simulation time (i.e., execution times of S_i, W_i, R_i, and A_i are known beforehand), thus the value of the stride does not impact our model. We set $s = 1$ (or $m = n$), so every step always analyzes a frame.

Execution constraints. To ensure work conservation we define the following constraint: $\sum_{i=0}^{m} S_i = S$ (m is the number of produced frames). Obviously, we have identical constraints for R_i, A_i, and W_i. Similarly to the classic approach, we have the following precedence constraints for all i with $0 \le i \le m$:

$$S_i \rightarrow W_i \rightarrow R_i \rightarrow A_i. \tag{1}$$

Buffer constraints. The pipeline design of *in situ* workflows introduces new constraints. We consider a frame is analyzed right after it has been computed. This implies that for any given step i, the stage W_{i+1} can start if, and only if, the stage R_i has been completed. Formally, for all i with $0 \le i \le m$:

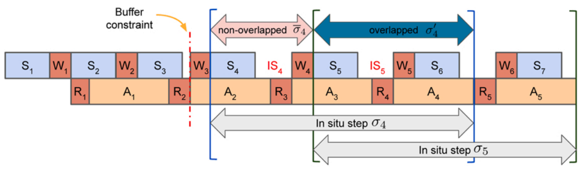
$$R_i \rightarrow W_{i+1}. \tag{2}$$

Eqs. (1) and (2) guarantee that we buffer at most one frame at a time (Fig. 3). Note that, this constraint can be relaxed such that up to k frames can be buffered at the same time as follows, $R_i \rightarrow W_{i+k}$, where $0 \le i \le m$ and $1 \le k \le m$. In this work, we only consider the case $k = 1$ (red arrows in Fig. 4).

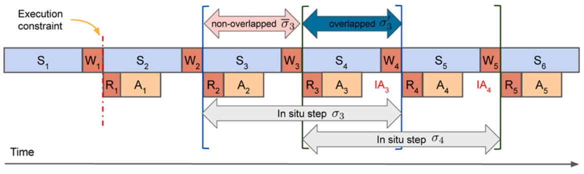
Idle stages. Due to the above constraints, the different stages are tightly-coupled (i.e., R_i and A_i stages must wait S_i and W_i before starting their executions). Therefore, idle periods could arise during the execution (i.e., either the simulation or the analytics must wait for the other component). We can characterize two different scenarios, *Idle Simulation* and *Idle Analyzer* in which idle time occurs. The former (Fig. 3(a)) occurs when analyzing a frame takes longer to complete compared to a simulation cycle (i.e., $S_i + W_i > R_i + A_i$). The later (Fig. 3(b)) occurs when the simulation component takes longer to execute (i.e., $S_i + W_i < R_i + A_i$). Fig. 4 provides a detailed overview of the dependencies among the different stages. Note that, the concept of *in situ* step is defined and explained later in the paper.

Intuitively, we want to minimize the idle time on both sides. If the idle time is absent, then it means that we reach the idle-free scenario: $S_i + W_i = R_i + A_i$. To ease the characterization of these idle periods, we introduce two idle stages, one per component. Let I_i^S and I_i^A be, respectively, the idle time occurring in the simulation and in the analysis component for the step i. These two stages represent the idle time in both components, therefore the precedence constraint defined in Eq. (1) results in:

$$S_i \rightarrow I_i^S \rightarrow W_i \rightarrow R_i \rightarrow A_i \rightarrow I_i^A. \tag{3}$$



(a) *Idle Simulation* scenario when $S_* + W_* < R_* + A_*$



(b) *Idle Analyzer* scenario when $S_* + W_* > R_* + A_*$

Fig. 3. Two different execution scenarios for *in situ* workflow execution.

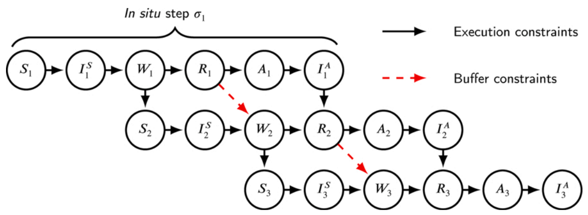


Fig. 4. Dependency constraints within and across *in situ* steps with $n = m = 3$. (For interpretation of the references to color in this figure citation, the reader is referred to the web version of this article.)

4.2. Consistency across steps

This work is supported by the hypothesis that every execution of *in situ* workflows under the above constraints will reach a consistent state after a finite number of warming-up steps. Thus, the time spent on each stage within an iteration can be considered constant over iterations. Formally, there exists j where $0 \le j < m$ such that for all i where $j \le i \le m$, we have $S_i = S_j$. The same holds for each stage, W_i, R_i, A_i, I_i^S, and, I_i^A. This hypothesis is confirmed in Section 5.3, and in practice, we observe that the cost of these non-consistent steps is negligible. Our experiments showed that, on average, $j \le 3$ for one hundred steps ($m = 100$). Therefore, we ignore the warming steps and we consider $j = 0$. For the sake of simplicity, we generalize *in situ* consistency behavior by denoting $S_* = S_i$ for all $i \ge j$. We also have similar notations for R_*, A_*, I_*^S, I_*^A and W_*. This hypothesis allows us to predict the performance of a given *in situ* workflow by monitoring a subset of steps, instead of the whole workflow. From the two constraints defined by Eqs. (3) and (2), and our hypothesis, we define:

$$S_* + I_*^S + W_* = R_* + A_* + I_*^A. \tag{4}$$

The Idle Simulation scenario is when $I_*^A = 0$, and $I_*^S = 0$ for Idle Analyzer scenario. Let I_* be the total idle time for an *in situ* step, using Eq. (4) we derive:

$$I_* = I_*^S + I_*^A = \begin{cases} R_* + A_* - S_* - W_*, & \text{if } I_*^A = 0, \\ S_* + W_* - R_* - A_*, & \text{if } I_*^S = 0, \end{cases} \tag{5}$$

$$= |S_* + W_* - (R_* + A_*)| \tag{6}$$

Here, we could define the idle time a bit differently by saying that

$$I_*' = S_* + W_* - (R_* + A_*). \tag{7}$$

We can keep the sign for this indicator, and then define

$$\begin{aligned} I_*' > 0 &\quad \rightarrow \text{Idle Analysis}, I_*^A = I_*', \\ I_*' < 0 &\quad \rightarrow \text{Idle Simulation}, I_*^S = -I_*'. \end{aligned}$$

An example of these values is plotted in Fig. 10, along with the equilibrium point, which defines the configurable parameters at which the workflow execution switches from one scenario to another. The equilibrium point is determined at the point where $I_* = I_*' = 0$.

We assume the read/write operations are fast compared to the simulation and analysis stages, which is confirmed in Section 5.3, we can write:

$$I_*' \approx S_* - A_* \tag{8}$$

Now we take the stride s, which defines after how many simulation steps a frame is outputted to the analytics, into consideration for further analysis. Assuming we know the wall-clock time t_S for one simulation step (using the output of running the simulation independently), we have $S_* = st_S$. The equilibrium point is attained when

$$st_S - A_* = 0 \tag{9}$$

from which we can deduce, for example, given some resource allocation, to minimize the idle time, theoretically, we should set the stride approximately to

$$\overline{s} = A_*/t_S. \tag{10}$$

In practice, the equilibrium point could slightly change from this optimal value due to the resource contention of co-located *in situ* components, which is discussed in detail in Section 6.

4.3. In situ step

The challenge behind *in situ* workflows evaluation lies in collecting global information from multiple components (in our case, the simulation and the analytics) and use this information to derivate meaningful characteristics about the execution. The complexity of such a task is correlated to the number of steps the workflow is running and the number of components involved. By leveraging the consistency hypothesis in Eq. (4), we propose to alleviate this cost by proposing a metric that does not require data from all steps. The keystone of our approach is the concept of *in situ* step. Based on Eq. (3), the *in situ* step σ_i is determined by the timespan between the beginning of S_i and the end of I_i^A. The *in situ* step concept helps us to manipulate all the stages of a given step as one consistent task executing across components that can potentially run on different machines.

Different *in situ* steps overlap each other, so we need to distinguish the part that is overlapped ($\sigma_i^{'}$) from the other part ($\overline{\sigma}_i$). Thus, $\sigma_i = \overline{\sigma}_i + \sigma_i^{'}$. For example, in Fig. 3(a), to compute the time elapsed between the start of σ_4 and the end of σ_5, we need to sum the two steps and remove the overlapped execution time $\sigma_4^{'}$. Thus, we obtain $\sigma_4 + \sigma_5 - \sigma_4^{'}$. This simple example will give us the intuition behind the makespan computation in Section 4.4.

The consistency hypothesis insures consistency across *in situ* steps. We denote σ_* as the consistent *in situ* step (i.e., $\forall\, i,\ \sigma_i = \sigma_*$), while $\sigma_*^{'}$ and $\overline{\sigma}_*$ indicate, respectively, the overlapped and the non-overlapped part of two consecutive *in situ* steps. Thus, $\sigma_* = \overline{\sigma}_* + \sigma_*^{'}$. To calculate the makespan, we want to compute the non-overlapped step $\overline{\sigma}_*$. As shown in Fig. 3, the non-overlapped period $\overline{\sigma}_*$ is the aggregation of all stages belonging to one single component in an *in situ* step:

$$\overline{\sigma}_* = \begin{cases} S_* + I_*^S + W_*, & \text{if } I_*^S \geq 0, \\ R_* + A_* + I_*^A, & \text{if } I_*^A \geq 0. \end{cases} \tag{11}$$

We have two scenarios, if $I_*^S = 0, I_*^A \geq 0$ (*Idle Analyzer*) then $\overline{\sigma}_* = S_* + W_*$, otherwise $\overline{\sigma}_* = R_* + A_*$. Hence,

$$\overline{\sigma}_* = max(S_* + W_*, R_* + A_*). \tag{12}$$

4.4. Makespan

A rough estimation of the makespan of such workflow would be the sum of the execution time for all the stages (i.e., sum up the *m in situ* steps σ_i). But, recall that *in situ* steps interleave with each other, so we need to subtract the overlapped parts:

$$\text{MAKESPAN} = m\sigma_* - \sigma_*^{'}\left(m-1\right) = m\overline{\sigma}_* + \sigma_*^{'}. \tag{13}$$

From Eq. (13), for *m* large enough, the term $\sigma_*^{'}$ becomes negligible. Since *in situ* workflows are executed with a large number of iterations, then $\text{Makespan} = m\overline{\sigma}_*$ (recall that $\sigma_* = \overline{\sigma}_* + \sigma_*^{'}$). This observation indicates that the non-overlapped part of an *in situ* step is enough to characterize a periodic *in situ* workflow. From Eq. (11), minimizing the makespan is equivalent to reducing the idle time to zero, which confirms the smallest makespan occurs at the equilibrium point (i.e., when the idle time is equal to zero). Therefore, the *in situ* execution is necessary to be driven to the equilibrium point. Using our framework and these observations, we further define a metric to estimate the efficiency of a workflow.

4.5. In situ efficiency

Based on our *in situ* execution model, we propose a novel metric to evaluate resource usage efficiency *E* of an *in situ* workflow. We define efficiency as the time wasted during execution—i.e., idle times I_*^S and I_*^A. This metric considers all the components (simulation and the analysis) for evaluating in situ workflows:

$$E = 1 - \frac{I_*}{\overline{\sigma}_*} = 1 - \frac{|S_* + W_* - (R_* + A_*)|}{max(S_* + W_*, R_* + A_*)}. \tag{14}$$

This efficiency metric allows for performance comparison between different *in situ* runs with different configurations. By examining only one non-overlapped *in situ* step, we provide a lightweight approach to observe behavior from multiple components running concurrently in an *in situ* workflow. This metric can be used as an indicator to determine how far the *in situ* execution is from the equilibrium point, where $I_* = 0$ or $E = 1$. We then get an idea of how to adjust the parameters to approach the equilibrium point, at which the makespan is minimized (see Section 4.4). Note that, this model and the efficiency metric can be easily generalized to any number of components.

5. Molecular dynamics synthetic workflow

MD is one of the most popular scientific applications executing on modern HPC systems. MD simulations reproduce the time evolution of molecular systems at a given temperature and pressure by iteratively computing inter-atomic forces and moving atoms over a short time step. The resulting trajectories allow scientists to understand molecular mechanisms and conformations. In particular, a trajectory is a series of *frames*, i.e., sets of atomic positions saved at fixed intervals of time. The *stride* is the number of time steps between frames considered for storage or further *in situ* analysis. For example in our framework, for a simulation with 100 steps and a stride of 20, only 5 frames will be sent by the simulation to the analysis component. Since trajectories are high-dimensional objects and many atomic motions such as high-frequency thermal fluctuations are usually of no interest, scientists use well-chosen collective variables (CVs) to capture important molecular motions. Technically, a CV is defined as a function of the atomic coordinates in one frame. Reduced to time series of a small number of such CVs, simulated molecular processes are much more amenable to interpretation and further analysis. A CV can be as simple as the distance between two atoms, or can involve complex mathematical operations on a large number of atoms. An example of a complex CV that we will use in this work is the largest eigenvalue of the bipartite matrix (LEBM). Given two amino acid segments *A* and *B*, if d_{ij} is the Euclidean distance between C_α atoms *i* and *j*, then the symmetric bipartite matrix $B_{\mathrm{AB}} = \left[b_{\mathrm{ij}}\right]$ is defined as follows:

$$b_{\mathrm{ij}} = \begin{cases} d_{\mathrm{ij}}, & \text{if } i \in A \text{ and } j \in B, \\ d_{\mathrm{ij}}, & \text{if } i \in B \text{ and } j \in A, \\ 0, & \text{otherwise}. \end{cases} \tag{15}$$

Note that B_{AB} is symmetric and has zeroes in its diagonal. Johnston et al. [19] showed that the largest eigenvalue of B_{AB} is an efficient proxy to monitor changes in the conformation of *A* relative to *B*.

5.1. Workflow description

In order to study the complex behavior resulting from coupling a MD simulation and with an analysis component in the previously discussed parameter space, we have designed a synthetic *in situ* MD workflow (Fig. 5). The Synthetic Simulation component extracts frames from previously computed MD trajectories instead of performing an actual,

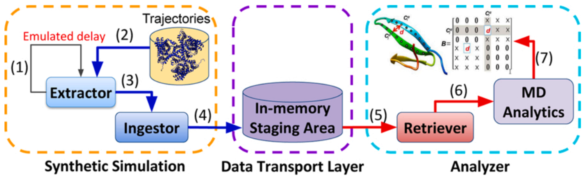


Fig. 5. Synthetic Workflow: the Extractor (1) sleeps during the emulated delay, then (2) extracts a snapshot of atomic states from existing trajectories and (3) stores it into a synthetic frame. The Ingestor (4) serializes the frame as a chunk and stages it in memory, then the Retriever (5) gets the chunk from the DTL and deserializes it into a frame. Eventually, the MD Analytics performs certain analysis algorithm on that frame.

compute-intensive MD simulation. This synthetic workflow is an implementation of the general and abstract software architecture proposed in Section 3.

Synthetic Simulation. The *Synthetic Simulation* emulates the process of a real MD simulation by extracting frames from trajectories generated previously by an MD simulation engine. The Synthetic Simulation enables us to tune and manage many simulation parameters (discussed in detail in Section 5.2) including the number of atoms and strides, which helps the Synthetic Simulation mimic the behavior of real molecular dynamics simulation. Note that, since the Synthetic Simulation does not emulate the computation part of the real MD simulation, it mimics the behavior of the I/O processes of the simulation. Thus, we define the *emulated simulation delay*, which is the period of time corresponding to the computation time in the real MD simulation. In order to estimate such delay for emulating the simulation time for a given stride and number of atoms, we use recent benchmarking results from the literature obtained by running the well-known NAMD [22] and Gromacs [24] MD engines. We considered the benchmarking performance for five practical system sizes of 81K, 2M, 12M atoms from Gromacs [24] and 21M, 224M atoms from NAMD [22] to interpolate to the simulation performance with the desired number of atoms (Fig. 6(a)). The interpolated value is then multiplied by the stride to obtain the delay (i.e., a function of both the number of atoms and the stride).

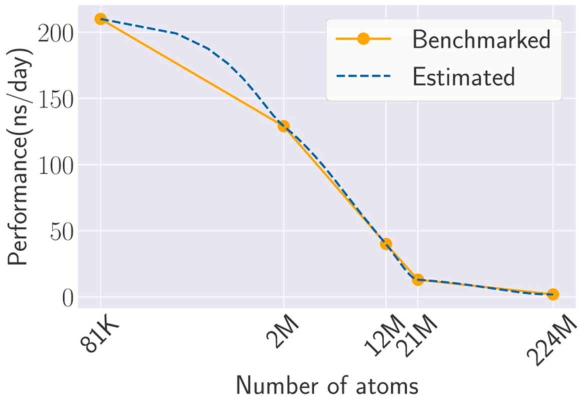
In Section 5.2, we run the synthetic workflow with 200 K, 400 K, 800 K, 1.6 M, 3.2 M, and 6.4 M protein atoms. Fig. 6(b) shows the emulated simulation delay when varying the stride for different numbers of atoms. The stride varying between 4 K and 16 K delivers a wide range of emulated simulation delays, up to 40 s.

Data Transport Layer (DTL). The *DTL Server* leverages DataSpaces [12] to deploy an in memory staging area for coupling data between the Synthetic Simulation and the Analyzer. DataSpaces follows the publish/subscribe model in terms of data flow, and the client/server paradigm in terms of control flow. The workflow system has to manage a DataSpaces server to manage data requests, keep metadata, and create in memory data objects.

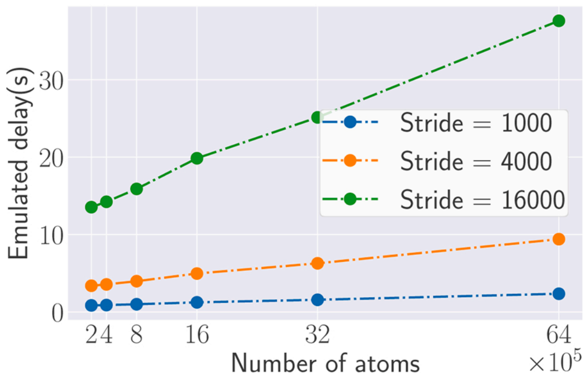
Analyzer. The *Analyzer* plays the role of the analytics component in the synthetic *in situ* workflow. More specifically, the *Retriever* subscribes to a chunk from the in memory staging area and deserializes it into a frame. The MD Analytics then performs a given type of analysis on this frame. Recall that, in our model, only one frame at a time can be store by the DTL (see Fig. 4). We leverage DataSpaces built-in locks to ensure that a writing operation to the in memory staging area can only happen when the reading operation of the previous step is complete (constraint model by Eq. (2)). Thus, the Analyzer is instructed by DataSpaces to wait for the next chunk available in the in memory staging area. Once a chunk has been received and is being processed, the Synthetic Simulation can send another chunk to the Analyzer.

5.2. *Experimental setups*

For our experiments we use Tellico (UTK), an IBM POWER9 system that includes four 32-core nodes (2 compute nodes) with 256 GB RAM each. Each compute node is connected through an InfiniBand interconnect network. Since the Synthetic Simulation only emulates the I/O operations of an MD simulation without replicating the actual computation, resource contention is not expected to produce the disparity in execution performance between different component placements. Moreover, the main contribution of the Synthetic Simulation is its ability of mimicking the actions of a real simulation engine with fewer resource requirements. For these reasons, we leverage this synthetic workflow to (i) validate the accuracy of the proposed *in situ* metrics; and (ii) characterize the behavior of coupling a simulation with a variety of system sizes with an *in situ* analytics. The Synthetic Simulation runs on one physical core on a single compute node as it mimics the behavior of a real simulation. On the other hand, the Analyzer and the DataSpaces server are co-located on another dedicated node. Particularly, the Analyzer computes bipartite matrices (see Section 5) using multiple parallel processes, which improves CV calculation efficiency. After experimenting with different numbers of Analyzer processes, we fixed that number at 16 processes (number of cores of an IBM AC922) to reach a good speed up and to fit the entire Analyzer within one compute node.



(a) Interpolated MD performance



(b) Emulated simulation delay

Fig. 6. MD benchmarking results from the literature obtained by using 512 NVIDIA K20X GPUs. The results are interpolated to obtain the (a) estimated performance and then combined with the stride to synthesize the (b) emulated simulation delay.

Parameter space. Table 3 describes the parameters used in the experiments. For the Synthetic Simulation, we study the impact of the number of atoms (the size of the system) and the stride (the frequency at which the Synthetic Simulation component sends a frame to the Analyzer through the DTL). We consider a constant number of 100 frames to be analyzed due to the time constraint and the consistency in the behavior between *in situ* steps. For the DTL, we use the staging method DATASPACES for all the experiments, in which the staging area

Table 3
Parameters used in the experiments.

Parameter	Description	Values used in the experiments
Synthetic simulation		
#atoms	Number of atoms	$[2\times10^5, 4\times10^5, 8\times10^5, 16\times10^5, 32\times10^5, 64\times10^5]$
#strides	Stride	[1000, 4000, 16,000]
#frames	Number of frames	100
Data transport layer		
SM	Staging method	DATASPACES
Analyzer		
CV	Collective variable	LEBM
lsegment	Length of segment pairs	16

is in the main memory of the node assigned to the DataSpaces server. For the Analyzer, we choose to calculate a compute-intensive set of CVs (LEBM, Section 5) for each possible pair of non-overlapping segments of length 16. If there are n amino acids (alpha amino acids) in the system, there are $N = \mathrm{floor}(n/16)$ segments, which amounts to $N(N-1)/2$ LEBM calculations ($\mathcal{O}(n^2)$). To fairly interpret the complexity of this analysis algorithm related to the system size, we manipulate the number of amino acids to be proportional to the number of atoms. For example, the system of 400 K atoms yields 2 fold larger number of segments compared to the system of 200 K atoms. Fig. 7 illustrates the LEBM's runtime with respect to the number of atoms.

We leverage user-defined events to collect the proposed metrics using TAU [27] (Section 3). We focus on two different levels of information, the workflow level and the *in situ* step level (the time taken by the workflow to compute and analyze one frame).

At the *in situ* step level, each value is averaged over three runs for each step. At the workflow level, each value is averaged over all *in situ* steps at steady-state and then averaged over the three runs. We also depict the standard deviation to assess the statistical significance of the results. There are two levels of statistical error: for averages across the 3 trials at the *in situ* step level, and for averages over 94 *in situ* steps (excluding the three first steps and the three last steps) in each run at the workflow level.

5.3. In situ step characterization

We study the correlation of individual stages in each *in situ* step. Due to lack of space, the discussion is limited to a subset of the parameter space as the representative of two characterized idle-based scenarios. Fig. 8 shows the execution time per step for each component with stride values of 1000 and 4000 steps. Confirming the consistency hypothesis across steps discussed in Section 4.2, we observe that the execution time per step is nearly constant, except for a few warm-up and wrap-up steps. Fig. 8(a) falls under the *Idle Simulation* (IS) scenario, as the I_i^S stage only appears in the Synthetic Simulation step. Similarly in Fig. 8(b), we observe the *Idle Analyzer* (IA) scenario because of the presence of I_i^A.

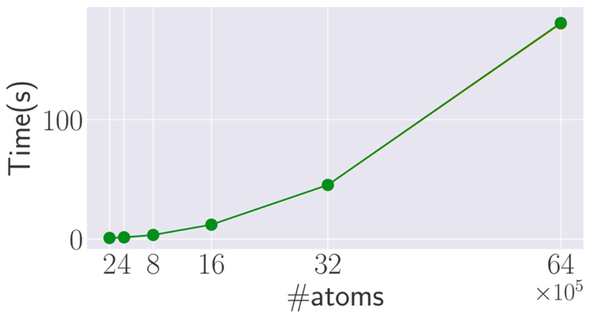
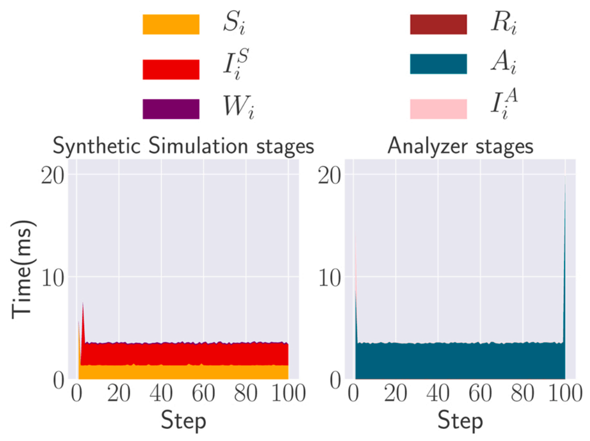
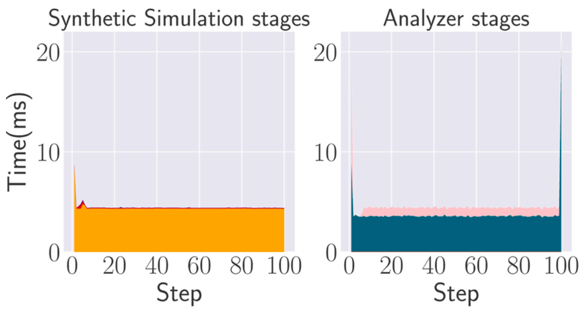


Fig. 7. Execution time of LEBM on 16 cores, and using a segment length of 16. The fraction of alpha-amino acids in the entire system is equal to 0.00469.



(a) stride = 1000 with 8×10^5 atoms



(b) stride = 4000 with 8×10^5 atoms

Fig. 8. Execution time per step for each component. The Synthetic Simulation stages are on the left and the Analyzer stages are on the right (lower is better).

These findings verify the existence of the two idle-based scenarios discussed in Section 4.1. Since both the Synthetic Simulation and Analyzer are nearly synchronized, we also underline that the execution time of a single step for each component is equal to each other. This information confirms the property of *in situ* workflows in Eq. (4). Overall, we can observe that the I/O stages (W_i and R_i) take an insignificant portion of time compared to the full step. This negligible overhead verifies the advantage of leveraging in-memory staging for exchanging frames between coupled components.

5.4. Idle time observation

By examining the total idle time, we study the impact of the number of atoms, stride on the performance of the entire workflow and each component for different scenarios.

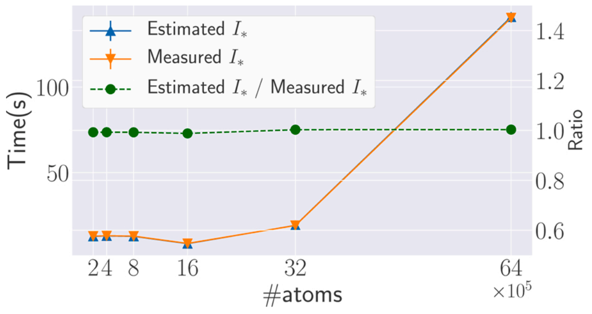


Fig. 9. Left *y*-axis: total idle time I_* using an *helper-core* placement at stride 16,000 (the lower the better). *Estimated* I_* is estimated from $|S_* + W_* - (R_* + A_*)|$, and *Measured* I_* is measured from $I_*^S + I_*^A$. Right *y*-axis: ratio *Estimated* I_* / *Measured* I_* (the closest to 1 the better).

Accuracy of estimated idle time. For different system sizes, Fig. 9 demonstrates the similarity between *Measured* I_* that is the measured idle time in one *in situ* step and *Estimated* I_*, which is the idle time estimation computed using Equation (5). The ratio between *Estimated* and *Measured* idle time is close to 1, confirming the accuracy of Eq. (5) to estimate the idle time I_* for each *in situ* step, which allows us to apply this relationship to identify the execution scenario that the workflow is following.

Execution scenarios. Fig. 10shows that the workflow execution follows our model (Fig. 3). The blue regions in Fig. 10 represent the *Idle Simulation* scenario when $S_* + W_* < R_* + A_*$, and the yellow area indicates the *Idle Analyzer* scenario when $S_* + W_* > R_* + A_*$. While increasing the number of atoms, which increases the simulation time and the chunk size, the total idle time I_* decreases in the *Idle Simulation* scenario, and increases in the *Idle Analyzer* scenario. Every *in situ* step exhibits a similar pattern in which at a certain system size the workflow execution switches from one scenario to another. We notice that with larger stride, the equilibrium point occurs at larger system sizes. As the stride increases, the Synthetic Simulation sends frames to the Analyzer less often. Therefore, increasing the stride reduces the gap between $S_* + W_*$ and $R_* + A_*$, which also leads to an equilibrium point reached with a smaller number of atoms. With a stride of 4000, the equilibrium point occurs at $\#\text{atoms} = 8 \times 10^5$, but it occurs at $\#\text{atoms} = 16 \times 10^5$ with a stride of 16,000. At a stride of 1000, the execution follows the *Idle Simulation* scenario for all observed number of atoms and the equilibrium point cannot be reached in this range of system size.

5.5. Estimated makespan

The goal is to verify the assertion made by Eq. (13) stating the MAKESPAN of an *in situ* workflow can simply be expressed as the product of the number of steps and the time of one step $m\overline{\sigma}_*$. A typical MD simulation can easily feature $>10^7$ number of *in situ* steps, thus a metric requiring only a few steps to be accurate is interesting. Fig. 11 demonstrates the strength of our approach to estimate the MAKESPAN (maximum error ~5%) using our definition of *in situ* steps, in addition to the accuracy of our model. In *in situ* workflows run with a larger number of steps, monitoring the entire system increases the pressure and slows down the execution. Thus, without failures and external loads, only looking at a single non-overlapped step results in a scalable, accurate, and lightweight approach.

5.6. Resource usage efficiency

We utilize the efficiency metric E given by Eq. (14), to evaluate an *in situ* configuration within the objective to propose a metric that allows users to characterize *in situ* workflows. Fig. 12 shows that the efficiency E increases and reach a maximum in the *Idle Simulation* scenario, and decreases after this maximum in the *Idle Analyzer* scenario. Thus, an *in situ* run is most efficient at the equilibrium point, where $E \approx 1$. If a run is less efficient and classified as the *Idle Analyzer* scenario, it has more freedom to perform other analyses or increase the analysis algorithm's complexity. In the *Idle Simulation* scenario, the simulation is affordable to emulate the motions of a larger number of atoms to more efficiently use the resource sitting idle.

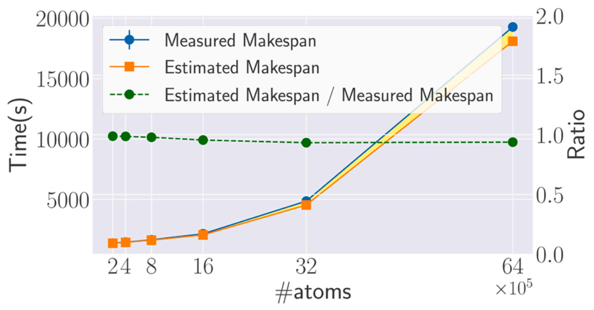


Fig. 11. MAKESPAN is estimated from $100\overline{\sigma}_*$ with stride 16,000, the yellow region represents the error. Ratio of *Estimated* MAKESPAN to *Measured* MAKESPAN uses the second y-axes on the right (close to 1 is better).

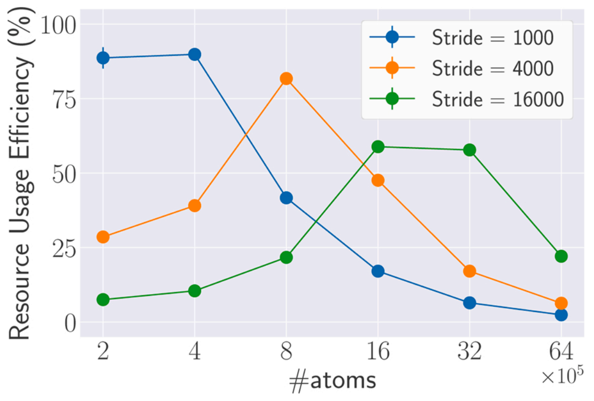


Fig. 12. Resource usage efficiency (higher is better).

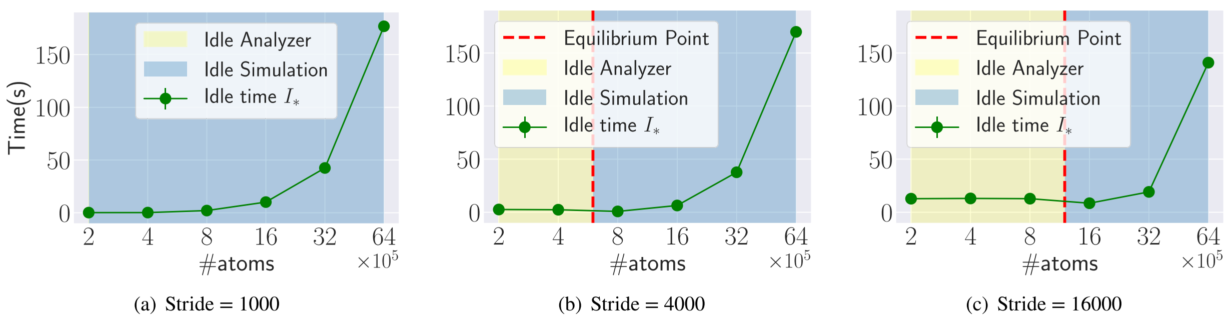
6. Molecular dynamics realistic workflow

In this section, in order to observe performance interference between applications running *in situ*, we replace the Synthetic Simulation by a high-performance molecular dynamics simulation engine. We use this realistic workflow to study the effect of different component placements on workflow execution characterization.

6.1. Workflow description

The simulation component in this practical workflow performs MD computation instead of sitting idle as the Synthetic Simulation does, thus it consumes memory that contends with the co-located Analyzer executing on the same resource.

Practical Simulation. The *Practical Simulation* (see Fig. 13) utilizes



(a) Stride = 1000 (b) Stride = 4000 (c) Stride = 16000

Fig. 10. Detailed idle time I_* for three component placements at different strides when varying the number of atoms (lower is better).

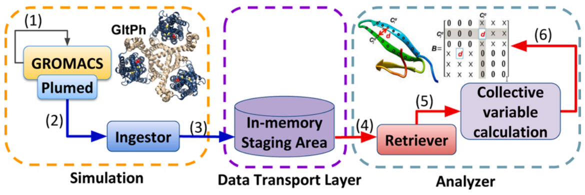


Fig. 13. Practical Workflow: GROMACS (1) simulates the motion of the atomic system in steps, where Plumed (2) interferes with every stride to update and gather new coordinates and store it into a frame. The Ingestor (3) serializes the frame as a chunk and stages it in memory, then the Retriever (5) gets the chunk from the DTL and deserializes it into a frame. Eventually, the MD Analytics performs the same analysis algorithm of CV calculation on that frame compared to Synthetic Workflow.

the MD software package GROMACS [24] to simulate biomolecular processes. GROMACS enables diverse levels of parallelism, i.e., multi-threading and process communication via message passing. However, this MD engine does not explicitly allow us to extract in-memory frames during the course of the run without manually intruding the source code. To offer a non-intrusive approach, we use Plumed [32] that intercepts function calls done by GROMACS periodically to get a snapshot of the system state in memory. An additional layer implemented by Plumed is placed on top of the corresponding simulation as an external library. Therefore, this Plumed kernel approach to obtaining in-memory frames is not restrictive to a specific simulation engine, but applicable to a variety of MD codes as long as Plumed provides support for being incorporated in such MD applications. In particular, a Plumed kernel function is called in every interval of time, which is determined by the stride, to collect atomic coordinates at the corresponding simulation step. Molecular positions are then serialized into an abstract chunk to be compatible with the data abstraction conducted by the interface of the Ingestor. Since the chunk is reachable by the Ingestor, the dataflow then acts similarly to the Synthetic Workflow.

Data Transport Layer and Analyzer. In this workflow, the DTL and the Analyzer remain the same at the synthetic workflow discussed in Section 5.

6.2. *Experimental setups*

In this experiment, we study two component placements: (i) *helper-core*—where the Synthetic Simulation, the DataSpaces server, and the Analyzer are co-located on the same compute node; and (ii) *in transit* where the Analyzer and the DataSpaces server are co-located on one node, and the Synthetic Simulation runs on a dedicated node. We use the same machine, Tellico, which is described in Section 5.2. The experimental plan is designed to yield baseline insights on a simulation coupled with an *in situ* analysis task in the context of ensemble workflows. An ensemble workflow is comprised of many small-scale simulations [23] that run independently of each other. Hence, insights from a single simulation-analysis integration will scale up to the entire workflow.

On the simulation side, we conduct a GROMACS run on 24 cores of a compute node, while the remaining cores of the node are assigned to the Analyzer and the DataSpaces server in the *helper-core* placement. Specifically, we run the analytics on 4 physical cores and set 1 core to execute a DataSpaces server. In contrast to the *helper-core* placement, the Analyzer and the DTL server reside in a separate node in the *in transit* placement. We keep the resources assigned for each component comparable in both placements to demonstrate the impact of such the component placement on the execution of an *in situ* workflow. The details of the parameter space used in this experiment are specified in Table 4.

For the Practical Simulation, we selected a medium-scale all-atom system that we used in a previous publication [3] on the molecular

Table 4
Parameters used in the experiments.

Parameter	Description	Values used
Practical simulation—Gltph system		
#atoms	Number of atoms	268,552
#strides	Stride	[1000, 2000, 3000, 4000, 5000]
#steps	Number of simulation steps	45,000
#frames	Number of frames	#steps / #strides
Data transport layer		
SM	Staging method	DATASPACES
Analyzer		
CV	Collective variable	LEBM
lsegment	Length of segment pairs	2
#threads	Number of threads	4
#repetitions	Number of times computing CV	10

mechanism of active neurotransmitter transport across the cellular membrane. That study focused on the GltPh transporter protein, which is an archaeal homolog of the human excitatory amino acid transporter (EAAT) family of proteins which are implicated in many neurological disorders and are responsible for permanent neurological damage after strokes. The 268,552-atom model system contains the GltPh transporter protein (three identical chains of 605 amino acids, X-ray structure from PDB entry 2NWX [8]) embedded in a lipid bilayer and surrounded by water molecules and a physiological concentration of Na^+ and Cl^- ions. Molecular interactions are parameterized with the CHARMM36 force-field [6] implemented in GROMACS [7], with standard simulation settings and a time steps of 2 fs.

To test our analysis workflow with GltPh, we explore different strides of 1000, 2000, 3000, 4000, 5000 time steps, at which Plumed generates in-memory frames for later processing. Thus for each #strides, the number of generated frames on the simulation side, which is also equivalent to the number of analyzed frames on the analysis side, is calculated as $\frac{\#\text{steps}}{\#\text{strides}}$. In this experimental setup, we vary the stride as a configurable parameter to find the equilibrium point. However, this parameter is not only restricted to the stride, we are always able to set the equilibrium point to different parameters. On the Analyzer side, the analysis kernel computes a set of 10 LEBM collective variables (Section 5) for each possible pair of non-overlapping segments of length 2. The complexity of this CV computation with respect to the predefined segment length is discussed in Section 5.2. For the DTL, the memory staging method DATASPACES is used for all subsequent experiments in this section, and data resides in the memory of the node where the DataSpaces server runs.

Due to the difficulty of linking TAU to Plumed, in this experiment, we manually inserted timers to collect performance data that is necessary for the proposed *in situ* metrics. Similar statistical methods (Section 5.2) are applied, so we can accumulate experimental error across both trials and *in situ* steps at the same time. We still eliminate the first three *in situ* steps and the last *in situ* step to assure *in situ* metrics are collected in the steady state where consistent behavior is observed across *in situ* steps.

6.3. *In situ step characterization over component placements*

We examine each *in situ* step over different placements of *in situ* tasks. Fig. 14 illustrates the time spent in each stage on both the simulation and the analysis side at stride 1000. Since the practical workflow satisfies dependency constraints within an *in situ* step and across the aforementioned stages, the behavior is observed to be approximately stable as expected in the steady state regime. The results are shown at stride 1000 only due to lack of space, but we note that the consistency is present for every given stride. This experiment confirms the applicability of our proposed *in situ* metrics. In addition, the *in transit* scheme appears to result in less bursty behavior in terms of execution compared to the *helper-core* scenario. The fluctuations observed when using *helper-core*

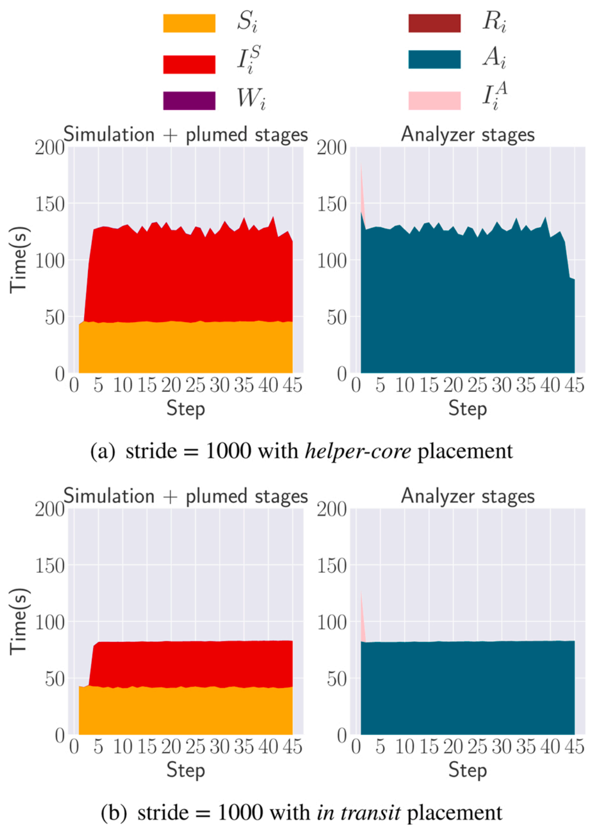


Fig. 14. Execution time per *in situ* step for each component with the *helper-core* and *in transit* placement. The Practical Simulation stages are on the left and the Analyzer stages are on the right (lower is better).

are due to resource contentions between co-located applications.

6.4. Makespan with different component placements

In Section 5.5, we have confirmed the assumption of consistency across steps and thus, our *in situ* metric, for the synthetic scenario and for one component placement. Here, we further estimate the Makespan from the non-overlapped step $\overline{\sigma}_*$ based on Eq. (13) using different component placements. In terms of execution scenarios, an *in situ* run is classified by the *Idle Simulation* and the *Idle Analyzer* in Section 4.1. In this experiment, we are able to determine which range of strides leads to which scenario as shown in Fig. 15. We define the *equilibrium point* as the inflexion point where the transition from *Idle Simulation* to *Idle Analyzer* happens (i.e., the equilibrium point corresponds to a perfect execution with zero idle time).

Fig. 15 compares the Makespan between the *helper-core* and *in transit* placement. At first glance, the estimated Makespan is close to the measured Makespan in both scenarios. The equilibrium point happens at stride 2000 to stride 3000 in the *in transit* scheme, whereas the equilibrium point of the *helper-core* placement occurs at larger stride (from stride 3000 to stride 4000). This finding confirms that the *in transit* configuration allows executing more frequent analyses at better efficiency than the *helper-core* does. In the *Idle Analyzer* case, there is no big difference in Makespan between the *helper-core* and the *in transit* case. Another way to state this is that component placement has more importance in the *Idle Simulation* scenario, which corresponds to the case when the analytics are performed at high frequency. Although the experiment is conducted on a single simulation coupled with an *in situ* analysis task, the trend observed here sets the foundation for scaling up to many simulations in the context of ensembles workflows.

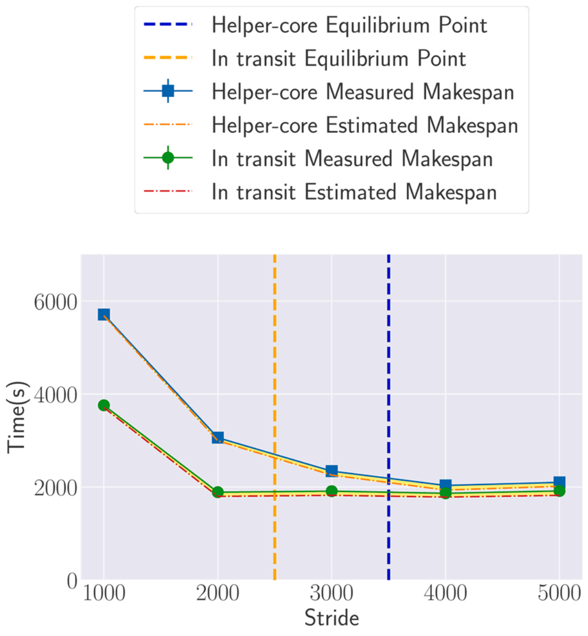


Fig. 15. Makespan is estimated from Eq. (13) over the *helper-core* and *in transit* component placement, the yellow region represents the error of the *Estimated* Makespan from the *Measured* Makespan.

6.5. Resource usage efficiency over component placements

As discussed in Section 5.4, evaluating the coupling performance of different component placements using the idle time in an *in situ* step is challenging due to the involvement of multiple concurrent tasks competing for computing resources and due to the large parameter space for each component. In this section, we leverage the efficiency metric E, Eq. (14), to determine how efficient an *in situ* run is with respect to a given configuration. Fig. 16 shows this efficiency value with different strides and over the *helper-core* and *in transit* placements. The comparison between two given *in situ* runs becomes straightforward using E as the indicator. The higher the E value is, the more efficient the *in situ* execution is, in terms of resource usage. The *helper-core* case has the best resource usage efficiency (~100%) at larger stride, or lower frequency

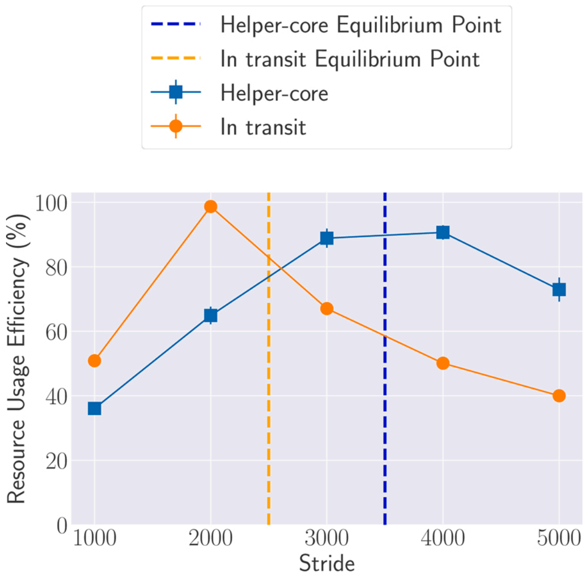


Fig. 16. Resource usage efficiency of the practical workflow over a variety of strides (higher is better).

of the Analyzer compared to the *in transit* case. Resource contention between co-located applications in the *helper-core* placement results in the efficiency degradation when performing the analytics at high frequency. This finding introduces a trade-off between computing resource cost and the analysis frequency in designing an *in situ* system. Finally, as expected, a run with a stride close to the equilibrium point gives a better resource usage efficiency.

7. Conclusions

In this study, we explored the challenges of evaluating next-generation *in situ* workflows. We have provided an analysis of *in situ* workflows by identifying a set of metrics that should be monitored to assess the performance of these workflows on HPC architectures. We have designed lightweight metric for the makespan and the computational efficiency of the workflow, based on behavior consistency across *in situ* steps under our constrained *in situ* execution model. We have validated the usefulness of these proposed metrics with a set of experiments using an *in situ* MD synthetic workflow. By using a realistic MD practical workflow, we have compared two different placements for the workflow components, a *helper-core* placement and an *in transit* placement in which the DTL server is co-located with different components. Under no resource constraint, by allocating dedicated nodes for the *in transit* analytics, the *in situ* coupling is allowed to perform the analysis more frequently. On the other hand, running the *helper-core* placement at the equilibrium point is targeted as the ideal scenario for optimizing resource utilization if those are limited.

Future work will study different models where the constraints are relaxed, for example where the workflow allows to buffer multiple frames in memory instead of one currently. We also plan to generalize the proposed framework's constraints to support more communication protocols, i.e., message-driven dataflow, multiple data transport paths, or another data transport layer. Another promising research line is to extend our theoretical framework to take into account multiple analysis methods, which is often the case for MD trajectory data. In this case, the time taken by the analysis could vary depending on the method used. Finally, arising from the necessity of more complex workflows to serve various *in situ* analysis requirements, performance evaluation of *in situ* workflows should be analyzed in the setting of an ensemble of workflows.

Authors' contribution

Tu Mai Anh Do: conceptualization, methodology, software, validation, formal analysis, investigation, data curation, writing—original draft, visualization. Loïc Pottier: conceptualization, methodology, validation, formal analysis, investigation, writing—original draft. Silvina Caíno-Lores: review & editing. Rafael Ferreira da Silva: writing—review & editing, funding acquisition. Michel A. Cuendet: writing—review & editing, funding acquisition. Harel Weinstein: review & editing, funding acquisition. Trilce Estrada: funding acquisition. Michela Taufer: review & editing, funding acquisition. Ewa Deelman: review & editing, supervision, funding acquisition.

Declaration of Competing Interest

The authors report no declarations of interest.

Acknowledgments

This work is funded by NSFcontracts #1741040, #1740990 and #1741057; and DOE contract #DE-SC0012636. We are grateful to IBM for the Shared University Research Award that supported the purchase of IBM Power9 system used in this paper. Extensive MD simulations for establishing conditions and the CV parameters utilized the resources (BIP109) of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract #DE-AC05-00OR22725. We would like to thank Sameer Shende, Nicholas Chaimov, Wyatt Spear from the TAU team and Melissa Romanus Abdelbaky, Philip Davis from the DataSpaces team for their help.

References

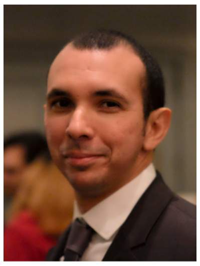
[1] L. Adhianto, et al., Hpctoolkit: tools for performance analysis of optimized parallel programs, Concurr. Comput.: Pract. Experience 22 (2010).
[2] A. Agelastos, et al., LDMS: A scalable infrastructure for continuous monitoring of large scale computing systems and applications, in: SC'14 (2014).
[3] N. Akyuz, et al., Transport domain unlocking sets the uptake rate of an aspartate transporter, Nature 518 (2015) 68–73.
[4] ASCR Workshop on In Situ Data Management. https://www.osti.gov/servlets/purl/1493245/.
[5] A.C. Bauer, et al., In situ methods, infrastructures, and applications on high performance computing platforms. Computer Graphics Forum, 2016.
[6] R.B. Best, et al., Optimization of the additive charmm all-atom protein force field targeting improved sampling of the backbone ϕ, ψ and side-chain χ_1 and χ_2 dihedral angles, J. Chem. Theory Comput. 8 (2012) 3257–3273.
[7] P. Bjelkmar, et al., Implementation of the CHARMM force field in GROMACS: analysis of protein stability effects from correction maps, virtual interaction sites, and water models, J. Chem. Theory Comput. 6 (2010) 459–466.
[8] O. Boudker, et al., Coupling substrate and ion binding to extracellular gate of a sodium-dependent aspartate transporter, Nature 445 (2007) 387–393.
[9] J.Y. Choi, et al., Coupling exascale multiphysics applications: methods and lessons learned. 2018 IEEE 14th International Conference on e-Science (e-Science) (2018).
[10] L. DeRose, et al., Cray performance analysis tools, Tools for High Performance Computing (2008) 191–199.
[11] T.M.A. Do, et al., A novel metric to evaluate in situ workflows. Computational Science – ICCS 2020, Springer International Publishing, Cham, 2020, pp. 538–553.
[12] C. Docan, et al., Dataspaces: an interaction and coordination framework for coupled simulation workflows, Cluster Comput. (2012).
[13] M. Dreher, et al., Decaf: Decoupled Dataflows for In Situ High-Performance Workflows, 2017, https://doi.org/10.2172/1372113.
[14] P. Fernando, et al., Nvstream: accelerating hpc workflows with nvram-based transport for streaming objects. Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing, ACM, New York, NY, USA, 2018, pp. 231–242.
[15] I. Foster, et al., Computing just what you need: Online data analysis and reduction at extreme scales, European Conference on Parallel Processing (2017).
[16] Y. Fu, et al., Performance analysis and optimization of in-situ integration of simulation with data analysis: zipping applications up. Proceedings of the 27th International Symposium on High-Performance Parallel and Distributed Computing, ACM, New York, NY, USA, 2018, pp. 192–205.
[17] D. Ghoshal, L. Ramakrishnan, Madats: Managing data on tiered storage for scientific workflows, Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing (2017) 41–52.
[18] R. Izadpanah, et al., Integrating low-latency analysis into HPC system monitoring. Proceedings of the 47th International Conference on Parallel Processing (ICPP), ACM, New York, NY, USA, 2018.
[19] T. Johnston, et al., In situ data analytics and indexing of protein trajectories, J. Comput. Chem. (2017).
[20] J.F. Lofstead, et al., Flexible io and integration for scientific codes through the adaptable io system (adios), 6th International Workshop on Challenges of Large Applications in Distributed Environments (2008).
[21] P. Malakar, et al., Scalable in situ analysis of molecular dynamics simulations. Proceedings of the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization, ACM, New York, NY, USA, 2017, pp. 1–6.
[22] https://www.ks.uiuc.edu/Research/namd/benchmarks/.
[23] J. Ossyra, et al., Porting adaptive ensemble molecular dynamics workflows to the summit supercomputer. High Performance Computing, Springer International Publishing, 2019.
[24] S. Páll, et al., Tackling exascale software challenges in molecular dynamics simulations with gromacs. Solving Software Challenges for Exascale, Springer International Publishing, Cham, 2015.
[25] L. Pouchard, et al., Prescriptive provenance for streaming analysis of workflows at scale. New York Scientific Data Summit, 2018.
[26] D. Rogers, et al., Data Co-Processing for Extreme Scale Analysis Level II ASC Milestone (4745). Technical Report, Sandia National Lab. (SNL-NM), Albuquerque, NM, United States, 2013.
[27] S.S. Shende, A.D. Malony, The Tau parallel performance system, Int. J. High Perform. Comput. Appl. 20 (2006) 287–311.
[28] R. Ferreira da Silva, et al., A characterization of workflow management systems for extreme-scale applications, Future Gen. Comput. Syst. 75 (2017).
[29] R. Ferreira da Silva, et al., On the use of burst buffers for accelerating data-intensive scientific workflows, 12th Workshop on Workflows in Support of Large-Scale Science (WORKS'17) (2017).
[30] I.J. Taylor, et al., Workflows for e-Science: Scientific Workflows for Grids, vol. 1, Springer, 2007.
[31] S. Thomas, et al., Characterizing in situ and in transit analytics of molecular dynamics simulations for next-generation supercomputers. 15th eScience, 2019.
[32] G.A. Tribello, et al., Plumed 2: new feathers for an old bird, Comput. Phys. Commun. 185 (2014) 604–613.

[33] J.S. Vetter, et al., Extreme Heterogeneity 2018 – Productive Computational Science in the Era of Extreme Heterogeneity: Report for DOE ASCR Workshop on Extreme Heterogeneity, LBNL, Berkeley, CA, United States, 2018. Technical Report.
[34] A. Vladimirov, R. Asai, MCDRAM as High-Bandwith Memory (HBM) in Knights Landing Processors: Developer's Guide, Colfax International, 2016. Technical Report.
[35] Y. Wang, et al., Smart: a mapreduce-like framework for in-situ scientific analytics, SC'15 (2015).
[36] C. Wood, et al., A scalable observation system for introspection and in situ analytics, 2016 5th Workshop on Extreme-Scale Programming Tools (ESPT) (2016).
[37] J.M. Wozniak, et al., Big data staging with mpi-io for interactive x-ray science, 2014 IEEE/ACM International Symposium on Big Data Computing (2014).
[38] F. Zhang, et al., In-memory staging and data-centric task placement for coupled scientific simulation workflows, CCPE 29 (2017).
[39] X. Zhang, et al., Wowmon: A machine learning-based profiler for self-adaptive instrumentation of scientific workflows, Proc. Comput. Sci. 80 (2016).
[40] H. Zou, et al., Flexanalytics: a flexible data analytics framework for big data applications with i/o performance improvement, Big Data Res. 1 (2014) 4–13.

Tu Mai Anh Do is a Computer Science Ph.D. student at the University of Southern California, and a Graduate Research Assistant in the Science Automation Technologies group at the USC Information Sciences Institute. His research covers several areas in Data Analytics and High Performance Computing. He received his B.E. in Computer Engineering from the Ho Chi Minh City University of Technology, Vietnam National University.

Loïc Pottier is a Computer Scientist at the USC Information Sciences Institute. He received his Ph.D. from École Normale Supérieure de Lyon (ENS Lyon), France, in 2018 where he worked on concurrent scheduling for High-Performance Computers (HPC). He was a postdoctoral researcher at the USC Information Sciences Institute in 2019 where he worked on scientific workflows optimization at scale. His recent research interests include scheduling techniques and algorithms for parallel systems and I/O management of large-scale scientific applications.

Silvina Caíno-Lores is a Post-Doctoral Research Associate in the University of Tennessee-Knoxville, as a member of the Global Computing Laboratory. She obtained her PhD in Computer Science and Technology in 2019 at the Carlos III University of Madrid (Spain). Her research interests include cloud computing, in-memory computing and storage, HPC scientific simulations, and data-centric paradigms. Her recent works and active collaborations focus on the area of convergence between HPC and Big Data analytics at the application and platform layers.

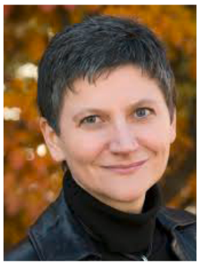


Rafael Ferreira da Silva is a Research Assistant Professor in the Department of Computer Science at University of Southern California, and a Research Lead at the USC Information Sciences Institute. His research focuses on the efficient execution of scientific workflows on heterogeneous distributed systems (including HPC and HTC), and modeling and simulation of parallel and distributed computing systems. Dr. Ferreira da Silva received his Ph.D. in Computer Science from INSA-Lyon, France, in 2013 (https://rafaelsilva.com).

Michel A. Cuendet is a Group Leader at the Precision Oncology Center of the Lausanne University Hospital in Switzerland, a Senior Scientist at the Swiss Institute of Bioinformatics, and a Visiting Research Assistant Professor in the Department of Physiology and Biophysics at Weill Cornell Medicine, New York, USA. Physicist by, training, Michel A. Cuendet is conducting research in areas including methodological developments for molecular dynamics simulations, modeling of large cellular receptors and transporter proteins, as well as data-centered and machine learning approaches for Precision Oncology.

Harel Weinstein, D.Sc. is the Maxwell Upson Professor of Physiology and Biophysics and Chairman of the Department of Physiology and Biophysics, and Founding Director of the Institute for Computational Biomedicine (ICB), a pioneering academic and research unit responsible for quantitative understandings of physiological function and disease, at Weill Cornell Medical College. A Tri-Institutional Professor, he holds appointments at Rockefeller University, Sloan-Kettering Institute and Cornell University. The Weinstein lab studies complex biomolecular systems with methods of molecular and computational biophysics, bioinformatics and mathematical modeling to learn about structural and dynamic mechanisms of cellular components. Biomedical end points include neurotransmission in health and disease, drug abuse mechanisms, and cancer.

Trilce Estrada is an associate professor of Computer Science at the University of New Mexico. Her research interests span the intersection of Machine Learning, Distributed Systems, Big Data, and their applications to interdisciplinary problems. She obtained her Ph.D. in computer science from University of Delaware, Masters' degree from the National Institute of Astrophysics Optics and Electronics (INAOE), and her undergraduate degree in Informatics from The University of Guadalajara, Mexico.



Michela Taufer (Senior Member, IEEE) is an ACM Distinguished Scientist and holds the Jack Dongarra professorship in high performance computing with the Department of Electrical Engineering and Computer Science, University of Tennessee Knoxville. Her research interests include high-performance computing, volunteer computing, scientific applications, scheduling and reproducibility challenges, and in situ data analytics. Dr. Taufer received her Ph.D. in Computer Science from the Swiss Federal Institute of Technology (ETH) in 2002.

Ewa Deelman is a Research Professor at the USC Computer Science Department and a Research Director at the USC Information Sciences Institute. Dr. Deelman's research interests include the design and exploration of collaborative, distributed scientific environments, with particular emphasis on workflow management as well as the management of large amounts of data and metadata. At ISI, Dr. Deelman is leading the Pegasus project, which designs and implements workflow mapping techniques for large-scale applications running in distributed environments. Pegasus is being used today in a number of scientific disciplines, enabling researches to formulate complex computations in a declarative way. Dr. Deelman received her Ph.D. in Computer Science from the RPI in 1997.