A Novel Metric to Evaluate In Situ Workflows

Tu Mai Anh Do[1], Loïc Pottier[1], Stephen Thomas[2], Rafael Ferreira da Silva[1], Michel A. Cuendet[3], Harel Weinstein[3], Trilce Estrada[4], Michela Taufer[2], and Ewa Deelman[1]

[1] USC Information Sciences Institute, Marina Del Rey, CA, USA
[2] University of Tennessee at Knoxville, Knoxville, TN, USA
[3] Weill Cornell Medicine, Cornell University, New York, NY, USA
[4] University of New Mexico, Albuquerque, NM, USA
{tudo,lpottier,rafsilva,deelman}@isi.edu, {sthoma99,mtaufer}@utk.edu,
{mac2109,haw2002}@med.cornell.edu, estrada@cs.unm.edu

Abstract Performance evaluation is crucial to understanding the behavior of scientific workflows and efficiently utilizing resources on high-performance computing architectures. In this study, we target an emerging type of workflow, called *in situ* workflows. Through an analysis of the state-of-the-art research on *in situ* workflows, we model a theoretical framework that helps characterize such workflows. We further propose a lightweight metric for assessing resource usage efficiency of an *in situ* workflow execution. By applying this metric to a simple, yet representative, synthetic workflow, we explore two possible scenarios (Idle Simulation and Idle Analyzer) for the execution of real *in situ* workflows. Experimental results show that there is no substantial difference in the performance of both the *in transit* placement (analytics on dedicated nodes) and the *helper-core* configuration (analytics co-allocated with simulation) on our target system.

Keywords: Scientific workflow · in situ model · molecular dynamics · high-performance computing.

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 [14]. 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 [13]. Storage bandwidth has also failed to keep pace with the rapid computational growth of modern processors due to the stagnancy of I/O advancements [7,16]. 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 [2]. Therefore, storing the entire output of scientific simulations on disk causes major bottlenecks in workflow performance. To reduce this disparity, 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 [2]. This accelerates simulation I/O by bypassing the file system, pipelining the analysis, and improving the overall workflow performance [4].

An *in situ* workflow describes a scientific workflow with multiple components (simulations with different parameters, visualization, analytics, etc.) running concurrently [4, 16], potentially coordinating their executions using the same allocated resources to minimize the cost of data movement [2]. 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 [7]. 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 [2]. 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). Throughout 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 can be directly translated into production *in situ* workflows. This work makes the following contributions:

1. We discuss practical challenges in evaluating next-generation workflows;
2. We define a non-exhaustive list of imperative metrics that need to be monitored for aiding the characterization of *in situ* workflows. We model a framework for *in situ* execution to formalize the iterative patterns in *in situ* workflows—we develop a lightweight approach that is beneficial when comparing the performance of configuration variations in an *in situ* system; and
3. We provide insights into the behaviors of *in situ* workflows by applying the proposed metric in characterizing an MD workflow.

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 [5, 12]. With the advent of *in situ* workflows [3], new monitoring and profiling ap-

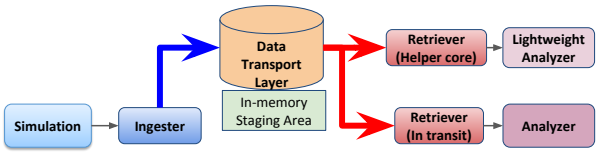


Figure 1. A general *in situ* workflow software architecture.

proaches targeting tightly-coupled workflows have been studied. LDMS [1] is a loosely-integrated scalable monitoring infrastructure that targets general large-scale applications that delivers a low-overhead distributed solution, in contrast to TAU [12], which provides a deeper understanding of the application at a higher computational cost. SOS [17] provides a distributed monitoring platform, conceptually similar to LDMS but specifically designed for online *in situ* characterization of HPC applications. ADIOS [9], the next-generation IO-stack, is built on top of many *in situ* data transport layers, e.g. DataSpaces [6]. Savannah [7], a workflow orchestrator, has been leveraged to bundle a coupled simulation with two main simulations, multiple analysis kernels, and a visualization service [4]. These works mainly focus on providing monitoring schemes for *in situ* workflows. This paper instead proposes a novel method to extract useful knowledge from the captured performance data.

In situ data management. FlexAnalytics [19] 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 [18] 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. 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*).

3 General *in situ* workflow architecture

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, several simulations coupled together. The *in situ* workflow architecture (Fig. 1) features three main components:

- A *simulation* component that performs MD computations and periodically generates data in the form of atomic coordinates.
- 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.

On the data path "simulation-to-analyzer" (1) the *ingester* ingests data from a certain data source and stores them in the Data Transport Layer (DTL) and the (2) *retriever*, in a reverse way, gets data from the DTL to perform further

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*

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 preprocess 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 [19]), 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. In this work, we use TAU to capture this information and we focus on how to use these data to characterize the *in situ* workflows.

4 *In situ* execution model

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. The simulation component (S) is the computational step that produces the data and (W) is the I/O stage that writes the produced data; The 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 [15]. The frequency of simulation steps to be analyzed is defined by *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 $m = \frac{n}{S}$. 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 produces 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 $0 \leq i \leq m$:

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

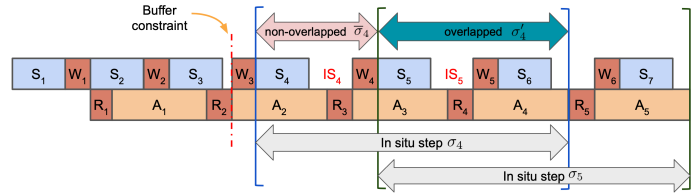
Buffer constraints. The pipeline design of *in situ* workflows introduces new constraints. We consider a frame is analyzed right after it has been computed. This implies that for any given step i, the stage W_{i+1} can start if, and only if, the stage R_i has been completed. Formally, for all i with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

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

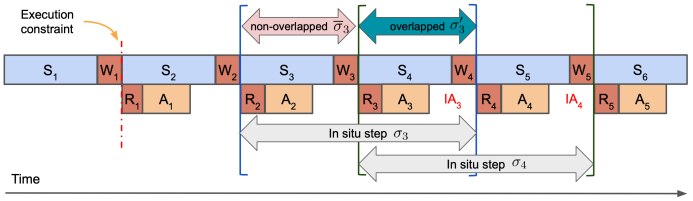
Eqs. 1 and 2 guarantee that we buffer at most one frame at a time (Fig. 2). Note that, this constraint can be relaxed such that up to k frames can be buffered at the same time as follows, $R_i \rightarrow W_{i+k}$, where $0 \leq i \leq m$ and $1 \leq k \leq m$. In this work, we only consider the case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

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 Analysis in which idle time occurs. The former (Fig. 2(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. 2(b)) occurs when the simulation component takes longer to execute (i.e., $S_i + W_i < R_i + A_i$). Fig. 3 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



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



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

Figure 2. Two different execution scenarios for *in situ* workflow execution.

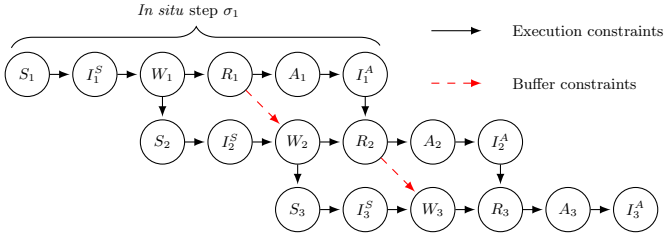


Figure 3. Dependency Constraints within and across *in situ* steps.

defined in Eq. 1 results in:

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

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 $0 \leq j < m$ such that for all i where $j \leq i \leq m$, 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 6, and in practice, we observe that the cost of these non-consistent steps is negligible. Our experiments showed that, on average, $j \leq 3$ 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 $i \geq j$. 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 Eq. 3 and Eq. 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:

$$I_* = I_*^S + I_*^A = \begin{cases} R_* + A_* - S_* - W_*, \text{ if } I_*^A = 0 \\ S_* + W_* - R_* - A_*, \text{ if } I_*^S = 0 \end{cases} = \left| S_* + W_* - (R_* + A_*) \right|. \tag{5}$$

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 (σ_i') from the other part ($\overline{\sigma}_i$). Thus, $\sigma_i = \overline{\sigma}_i + \sigma_i'$. For example, in Fig. 2(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 σ_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 σ_*' 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. 2, the non-overlapped period $\overline{\sigma}_*$ is the aggregation of all stages belonging to one single component in an *in situ* step: $S_* + I_*^S + W_*$ and $R_* + A_* + I_*^A$. Thus, we have two scenarios, if $I_*^S = 0$ then $\overline{\sigma}_* = S_* + W_*$, otherwise $\overline{\sigma}_* = R_* + A_*$. Hence, $\overline{\sigma}_* = \max(S_* + W_*, R_* + A_*)$.

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:

$$\textsc{Makespan} = m\,\sigma_* - \sigma_*'\,(m-1) = m\,\overline{\sigma}_* + \sigma_*'. \tag{6}$$

From Eq. 6, for m large enough, the term σ'_* becomes negligible. Since *in situ* workflows are executed with a large number of iterations, then MAKESPAN $= m\,\overline{\sigma}_*$. This observation indicates that the non-overlapped part of an *in situ* step is enough to characterize a periodic *in situ* workflow. Using our framework and these observations, we 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:

$$E = 1 - \frac{I_*}{\overline{\sigma}_*} = 1 - \frac{|S_* + W_* - (R_* + A_*)|}{\max(S_* + W_*, R_* + A_*)}. \tag{7}$$

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. 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, 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. 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 $B_{AB} = [b_{ij}]$ is defined by $b_{ij} = d_{ij}$ if $i \in A$, $j \in B$ or $i \in B$, $j \in A$ and 0 otherwise. Johnston et al. [8] showed that the largest eigenvalue of B_{AB} is an efficient proxy to monitor changes in the conformation of A relative to B. To study complex behavior of coupling between the MD simulation and the analysis component in exhaustively discussed parameter space, we have designed a synthetic *in situ* MD workflow (Fig. 4). The Synthetic Simulation component extracts frames from previously computed

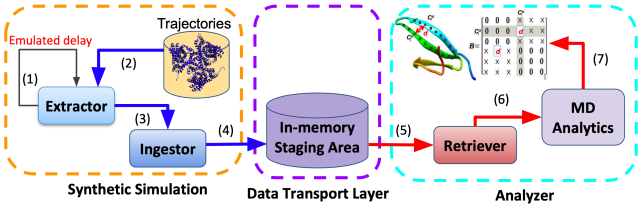


Figure 4. Synthetic Workflow: the Extractor (1) sleeps during the emulated delay, then (2) extracts a snapshot of atomic states from existing trajectories and (3) stores it into a synthetic frame. The Ingestor (4) serializes the frame as a chunk and stages it in memory, then the Retriever (5) gets the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

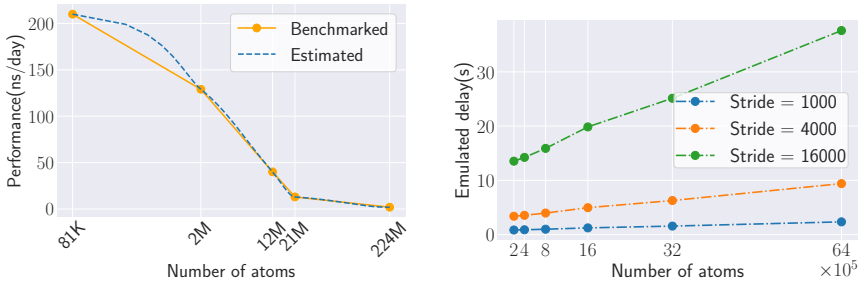
MD trajectories instead of performing an actual, compute-intensive MD simulation. This synthetic workflow is an implementation of the general and abstract software architecture proposed in Section 3.

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 6) 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 [10] and Gromacs [11] MD engines. We considered the benchmarking performance for five practical system sizes of 81K, 2M, 12M atoms from Gromacs [11] and 21M, 224M atoms from NAMD [10] to interpolate to the simulation performance with the desired number of atoms (Fig. 5(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 6, we run the synthetic workflow with 200K, 400K, 800K, 1.6M, 3.2M, and 6.4M protein atoms. Fig. 5(b) shows the emulated simulation delay when varying the stride for different numbers of atoms. The stride varying between 4–16K delivers a wide range of emulated simulation delay, up to 40s.

Data Transport Layer (DTL). The *DTL Server* leverages DataSpaces [6] 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



(a) Interpolated MD performance (b) Emulated simulation delay

Figure 5. 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.

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. 3). 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.

6 Experiments and Discussions

For our experiments we use Tellico (UTK), an IBM POWER9 system that includes four 32-core nodes (2 compute nodes) with 256GB RAM each. Each compute node is connected through an InfiniBand interconnect network. We target three component placements: (1) helper-core—where the Synthetic Simulation, the DataSpaces server, and the Analyzer are co-located on the same compute node; (2) *in transit S-S* where the Synthetic Simulation and the DataSpaces server are co-located on one node, and the Analyzer runs on another node; and (3) *in transit A-S* where the Analyzer and the DataSpaces server are co-located on one node, and the Synthetic Simulation runs on a dedicated node. Note that the Synthetic Simulation only runs on one physical core as it mimics the behavior of a real simulation. On the other hand, the Analyzer assigns bipartite matrices (see Section 5) to multiple processes, so the CV calculation is improved by parallel processing. Since we experimented to designate different numbers of Analyzer processes, we fix that number at 16 processes (number of cores of an IBM AC922) to attain good speed up and to fit the entire Analyzer in a compute node.

Table 2 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

Table 2. Parameters used in the experiments

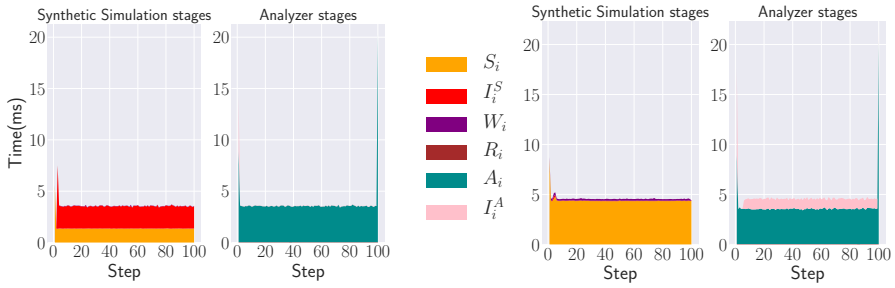
	Parameter	Description	Values used in the experiments
Synthetic simulation	#atoms	Number of atoms	$[2 \times 10^5, 4 \times 10^5, 8 \times 10^5, 16 \times 10^5, 32 \times 10^5, 64 \times 10^5]$
	#strides	Stride	[1000, 4000, 16000]
	#frames	Number of frames	100
Data transport layer	SM	Staging method	DATASPACES
Analyzer	CV	Collective variable	LEBM
	lsegment	Length of segment pairs	16

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. 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 $N = \text{floor}(n/16)$ 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.

We leverage user-defined events to collect the proposed metrics using TAU [12] (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-sate 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.

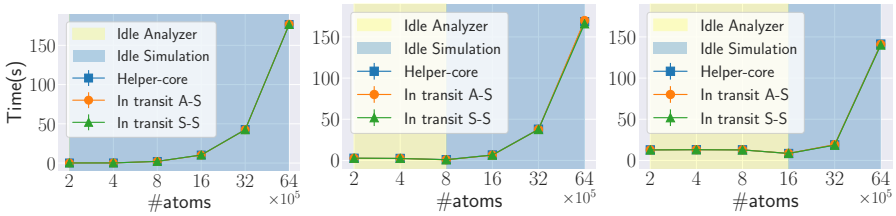
6.1 Experimental results

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. 6 shows the execution time per step for each component while varying the stride. 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. 6(a) falls under the *Idle Simulation* (IS) scenario, as the I_i^S stage only appears in the Synthetic Simulation step. Similarly in Fig. 6(b), we observe the *Idle Analyzer* (IA) scenario because of the presence of I_i^A. These findings verify the existence of 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* workflow in Eq. 4. Overall, we can observe that the I/O stages



(a) stride = 1000 with 8×10^5 atoms (b) stride = 4000 with 8×10^5 atoms

Figure 6. Execution time per step for each component with a *helper-core* placement (all component on the same node). The Synthetic Simulation stages are on the left and the Analyzer stages are on the right (lower is better).



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

Figure 7. Detailed idle time I_* for three component placements at different strides when varying the number of atoms (lower is better).

(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.

Execution scenarios. We study the impact of the number of atoms, stride, and components placement on the performance of the entire workflow and each component for different scenarios. Fig. 7 shows that the workflow execution follows our model (Fig. 2). While increasing the number of atoms, which increases the simulation time and the chunk size, the total idle time I_* decreases in the IS scenario, and increases in the IA 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 denote this point as the equilibrium point. We notice that with larger stride, the equilibrium point occurs at larger system sizes. The equilibrium point of the stride $4,000$ occurs at #atoms $= 8 \times 10^5$, but at #atoms $= 16 \times 10^5$ with the stride of $16,000$. In terms of component placement comparison, at first glance, there is no big difference in total idle time of the three placements (see Section 6.1).

Estimated makespan. The goal is to verify the assertion made by Eq. 6 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. 8 demonstrates the strength of our

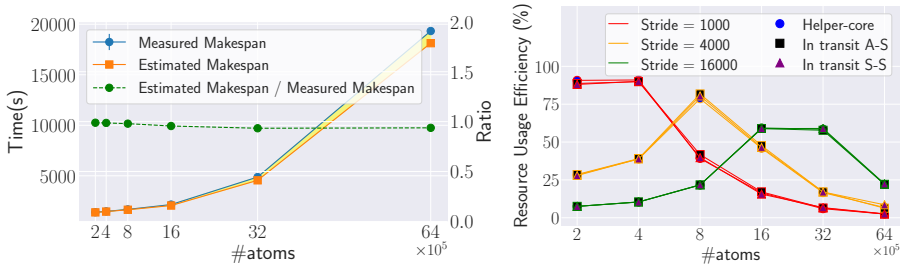


Figure 8. *Left:* MAKESPAN is estimated from $100\,\overline{\sigma}_*$ using the *helper-core* component placement with stride 16000, the yellow region represents the error. Ratio of *Estimated* MAKESPAN to *Measured* MAKESPAN uses the second y-axes on the right (close to 1 is better). *Right:* Resource usage efficiency (higher is better).

approach to estimate the MAKESPAN (maximum error $\sim 5\%$) using *in situ* steps and the accuracy of our model. *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.

Resource usage efficiency. We utilize the efficiency metric E given by Eq. 7, to evaluate an *in situ* configuration within the objective to propose a metric that allows users to characterize *in situ* workflows. Fig. 8-*right* shows that component placements have a small variation in resource efficiency. We infer that the placement of components is not a decisive factor in coupling performance. This result is consistent with the previous finding in Section 6.1. The efficiency E increases and reach a maximum in the IS scenario, and decreases after this maximum in the IA 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 IA scenario, it has more freedom to perform other analyses or increase the analysis algorithm's complexity. In the IS scenario, the simulation is affordable to larger system size.

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 a lightweight metric based on behavior consistency across *in situ* steps under constrained *in situ* execution model. We have validated the usefulness of this proposed metric with a set of experiments using an *in situ* MD synthetic workflow. We have compared three different placements for the workflow components, a *helper-core* placement and two *in transit* placements in which the DTL server co-locates with different components. 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. Another promising research line is to extend

our theoretical framework to take into account multiple analysis methods. In this case, the time taken by the analysis could vary regarding the method used.

Acknowledgments. This work is funded by NSF contracts #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.

References

1. Agelastos, A., et al.: LDMS: A scalable infrastructure for continuous monitoring of large scale computing systems and applications. In: SC'14 (2014)
2. ASCR Workshop on In Situ Data Management (2019)
3. Bauer, A.C., et al.: In situ methods, infrastructures, and applications on high performance computing platforms. In: Computer Graphics Forum. vol. 35 (2016)
4. Choi, J.Y., et al.: Coupling exascale multiphysics applications: Methods and lessons learned. 2018 IEEE 14th International Conference on e-Science (e-Science) (2018)
5. DeRose, L., et al.: Cray performance analysis tools. In: Tools for High Performance Computing, pp. 191–199. Springer (2008)
6. Docan, C., et al.: Dataspaces: an interaction and coordination framework for coupled simulation workflows. Cluster Computing (2012)
7. Foster, I., et al.: Computing just what you need: Online data analysis and reduction at extreme scales. In: European Conference on Parallel Processing. Springer (2017)
8. Johnston, T., et al.: In situ data analytics and indexing of protein trajectories. Journal of Computational Chemistry (2017)
9. Lofstead, J.F., et al.: Flexible io and integration for scientific codes through the adaptable io system (adios). In: 6th international workshop on Challenges of large applications in distributed environments (2008)
10. NAMD Performance. `https://www.ks.uiuc.edu/Research/namd/benchmarks/`
11. Páll, S., et al.: Tackling exascale software challenges in molecular dynamics simulations with gromacs. In: Solving Software Challenges for Exascale. Springer International Publishing, Cham (2015)
12. Shende, S.S., Malony, A.D.: The Tau parallel performance system. The International Journal of High Performance Computing Applications **20**(2), 287–311 (2006)
13. Ferreira da Silva, R., et al.: A characterization of workflow management systems for extreme-scale applications. Future Generation Computer Systems **75** (2017)
14. Taylor, I.J., et al.: Workflows for e-Science: scientific workflows for grids, vol. 1. Springer (2007)
15. Thomas, S., et al.: Characterizing in situ and in transit analytics of molecular dynamics simulations for next-generation supercomputers. In: 15th eScience (2019)
16. Vetter, J.S., et al.: Extreme Heterogeneity 2018 - Productive Computational Science in the Era of Extreme Heterogeneity: Report for DOE ASCR Workshop on Extreme Heterogeneity. Tech. rep., LBNL, Berkeley, CA (United States) (12 2018)
17. Wood, C., et al.: A scalable observation system for introspection and in situ analytics. In: 2016 5th Workshop on Extreme-Scale Programming Tools (ESPT) (2016)
18. Wozniak, J.M., et al.: Big data staging with mpi-io for interactive x-ray science. In: 2014 IEEE/ACM International Symposium on Big Data Computing (2014)
19. Zou, H., et al.: Flexanalytics: A flexible data analytics framework for big data applications with i/o performance improvement. Big Data Research **1**, 4 – 13 (2014)