Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

SPECIAL ISSUE PAPER

Performance Assessment of Ensembles of In Situ Workflows under Resource Constraints

Tu Mai Anh Do[1] | Loïc Pottier[1] | Rafael Ferreira da Silva[1,3] | Silvina Caíno-Lores[2] | Michela Taufer[2] | Ewa Deelman[1]

[1]Information Sciences Institute, University of Southern California, CA, USA
[2]University of Tennessee at Knoxville, TN, USA
[3]Oak Ridge National Laboratory, TN, USA

Correspondence
Tu Mai Anh Do, Information Sciences Institute, University of Southern California.
Email: tudo@isi.edu

Summary

Scientific breakthroughs in biomolecular methods and improvements in hardware technology have shifted from a long-running simulation to a large set of shorter simulations running simultaneously, called an ensemble. In an ensemble, simulations are usually coupled with analyses of data produced by the simulations. In situ methods can be used to analyze large volumes of data generated by scientific simulations at runtime (i.e., simulations and analyses are performed concurrently). In this work, we study the execution of ensemble-based simulations paired with in situ analyses using in-memory staging methods. Using an ensemble of molecular dynamics in situ workflows with multiple simulations and analyses, we first show that collecting traditional metrics such as makespan, instructions per cycle, memory usage, or cache miss ratio is not sufficient to characterize complex behaviors of ensembles. We propose a method to evaluate the performance of ensembles of workflows that captures multiple resource usage aspects: resource efficiency, resource allocation, and resource provisioning. Experimental results demonstrate that the proposed method can effectively distinguish the performance of different component placements in an ensemble with up to 32 ensemble members. By evaluating different co-location scenarios, our proposed performance indicators demonstrate benefits of co-locating simulation and coupled analyses within a compute node.

KEYWORDS:
Scientific workflow, Ensemble workflow, In situ model, Molecular dynamics, High-performance computing

1 | INTRODUCTION

Many simulations across scientific domains organize their computations into ensembles of workflows, the results of which are combined and analyzed, often using statistical analysis in order to gain insights and knowledge. Ensembles of workflows are composed of several inter-related workflows. These workflows typically have a similar structure, but they differ in their input data, number of tasks, and individual task sizes[1]. Workflow ensembles are used in molecular dynamics (MD) simulations, which compute the atomic states of a molecular system evolving over time by observing microscopic atomic interactions between atoms. For instance, studying the folding process of complex molecules (i.e., conformational transition) requires running large-scale

simulations with hundred of thousands of jobs to thoroughly explore feasible solutions in the configuration space. Such simulations require considerable computing time and resources that may grow exponentially with the size of the molecular system. These simulations are run in a concurrent fashion on high-performance computing (HPC) systems [2]. Ensemble-based simulation approaches may lead to more efficient sampling of the solution space. For instance, multiple-walker[3] employs multiple replicas of a molecular system, known as walkers, where each walker simultaneously explores the same free energy landscape to improve sampling performance. Generalized ensembles[4] allow sampling of a broader configuration space by partitioning simulation states into ensembles with optimal weights to perform a random walk in potential energy spaces.

Traditionally, MD simulations and the data analysis of their outputs (e.g., the molecular trajectories and energies across a simulation) are loosely coupled, where the analysis starts upon simulation completion. The coupling between the two components is typically done via the file system. However, because of the growing disparity between storage and computing capabilities in current leadership computers[5], post-processing of potentially large volumes of simulation data results in I/O bottlenecks[6] – writing data to storage during a simulation results into substantial slowdowns of the simulation itself, which experiences waiting for I/O. Furthermore, post-processing analysis does not allow runtime steering of the simulation to explore more promising configurations. In situ processing has emerged as an alternative paradigm to overcome such limitations[7]. Rather than post-processing data upon simulation completion, in situ methods allow scientists to process data during the runtime of the simulation by leveraging in-memory staging solutions such as DIMES[8], or fast local storage such as burst buffers[9] and performing the analysis concurrently. MD simulations, like many scientific simulations from diverse scientific domains, exhibit an iterative pattern that can benefit from the in situ paradigm (i.e., data generation and analysis can occur in concert). In this paper, we propose a solution to the efficient in situ processing problem in which simulations are coupled with analyses by staging simulation data into memory.

When running ensembles of in situ workflows, simulations and analyses can share the same resources, so that the data flowing between them can be efficiently communicated. However, resource sharing can lead to contentions and performance degradation due to interference[10]. In this paper, we present a method to characterize the execution of the workflow ensemble and to decide how the workflow components need to be placed within a system in order to optimize the overall workflow ensemble performance. To this end, we introduce a set of performance metrics that quantify the benefits of the co-location between components sharing the same computing allocation. We formalize the behavior of workflow ensembles into a theoretical framework and, then based on this framework we propose a method to evaluate resource usage, resource allocation, and resource provisioning for workflow ensembles. In addition to preliminary results from a recently published study[11], we validate the applicability of our proposed method on large-scale workflow ensembles, which have many simulations and in situ analyses coupled together using a variety of resource settings. Our contributions are as follows:

1. We introduce a set of comprehensive metrics that can characterize the overall workflow ensemble behavior at different levels of the application (task, workflow, and ensemble).
2. We propose a formal execution model to characterize in situ execution, which is then used to compute the efficiency of coupled components – this model lays out the foundation for our workflow ensemble performance framework.
3. We introduce novel performance indicators that allow us to assess the expected efficiency of a given configuration of a workflow ensemble in multiple resource perspectives.
4. We validate our proposed indicators using a realistic MD application executing on a leadership class system and empirically demonstrate the feasibility of using our methods in: (i) comparing the efficiency of various co-location scenarios with different resource configurations; and (ii) interpreting the performance of workflow ensembles that have a large number of simulations and analyses running in situ.

2 | Workflow ensemble

In this section, we conduct several experiments using a realistic use case of molecular dynamics ensembles executing on a large-scale HPC platform. We characterize the behavior of the ensemble use case using traditional metrics and discuss their limitations. The analysis of the obtained results demonstrates the need for new metrics that can accurately capture performance behaviors of ensemble-based computations. Based on these results, we develop new metrics that can accurately capture the ensemble behavior.

2.1 | Definitions

A *workflow ensemble* is a collection of inter-related *ensemble members/workflows* executing in parallel. Each ensemble member may be comprised of multiple *ensemble components* – a component can be a simulation or an analysis as is the case in our MD example (Figure 1). Note that even though a workflow ensemble can be comprised of parallel and sequential workflows, we can always group workflows (ensemble members) running in parallel into a workflow ensemble. We focus on the set of ensemble members running concurrently and starting their executions at the same time, to mimic how multiple MD simulations are executed simultaneously in ensemble methods[3,4]. In this work, we restrict ourselves to a single simulation per ensemble member. This simulation is coupled with at least one analysis component. We assume that ensemble members do not exchange information and are independent of each other (i.e., the analysis component of a given ensemble member only requires data generated by the simulation of that ensemble member[12]). The type of coupling is defined by the ensemble components. In our MD application, the simulation periodically writes out the data, which is read synchronously by the analyses. Although the simulation can compute while the analyses are reading the data, the simulation does not write any new data until the data from the previous iteration is fully consumed.

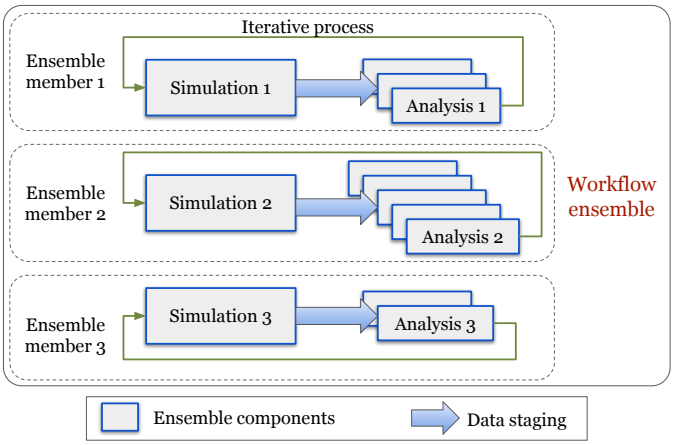

FIGURE 1 Ensemble of in situ workflows.

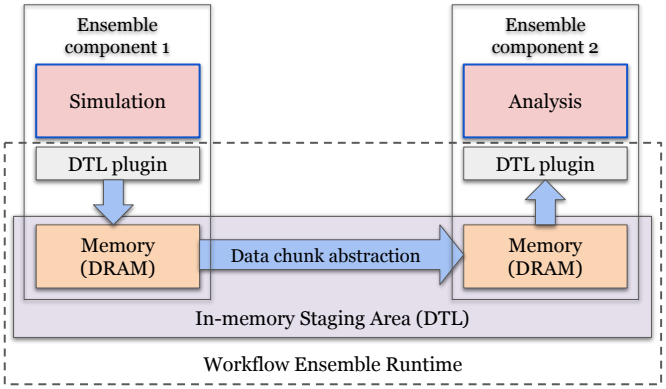

FIGURE 2 Overview of the proposed framework.

2.2 | Experimental Setup

In situ processing, combined with in-memory computing, has emerged as a solution to overcome I/O bottlenecks in large-scale systems, because moving data in memory rather than via the file system provides enhanced performance. However, using in situ processing often implies that the communicating components need to share a node on an HPC system (in case of a distributed memory architecture). This co-location can also lead to resource contention and thus reduce the benefit of in situ communications. In the context of workflow ensembles, a large number of components sharing resources may exacerbate resource contention. To measure the impact of resource contention, we monitor a set of traditional metrics (see Table 1) that are classified into three levels of granularity: (i) ensemble component, (ii) ensemble member/workflow, and (iii) workflow ensemble.

Metric	Description
Ensemble Component	
Execution time	Time spent in one component (e.g., simulation or analyses)
LLC miss ratio	Number of LLC misses / Number of LLC references
Memory intensity	Number of LLC misses / Number of instructions
Instructions per cycle	Number of instructions / Number of cycles
Ensemble Member	
Member makespan	Timespan between simulation start time and the latest analysis end time
Workflow Ensemble	
Ensemble makespan	Maximum makespan among all ensemble members in the workflow

TABLE 1 Set of metrics. (LLC stands for Last-level cache.)

At the ensemble component level, cache miss ratio and memory intensity[13] indicate the degree of resource contention; instructions per cycle shows the raw performance of the ensemble component. At the ensemble member level, we calculate the turnaround time (makespan) of each member, by computing the difference between the completion time of the latest analysis and the simulation start time. The ensemble makespan is defined as the maximum makespan among all ensemble members. (Recall that all members run concurrently and all simulations start simultaneously.)

Application. In this experiment, an ensemble member is comprised of a MD simulation coupled with analysis kernels using in situ processing. Specifically, the simulation simulates a medium-scale all-atom system containing the GltPh transporter protein[14]. Molecular interactions are implemented with GROMACS[15], with standard simulation settings at a time-step of 2 femtoseconds. The simulation periodically sends in-memory generated frames, i.e. atomic positions, to the analyses coupled with it. In our application, the analysis computes the largest eigenvalue of bipartite matrices[16,17] as a collective variable[18] of the frames. This captures molecular motions of the system. The frequency at which data is sent for analysis is determined by the *stride*, which represents the number of simulation steps computed before a frame is generated.

Workflow ensemble runtime. For our experiments, we developed a runtime system (Figure 2) that manages the execution of workflow ensembles on a target HPC platform. This runtime includes two main components: (i) a data transport layer (DTL), and (ii) a DTL plugin. The former represents a variety of storage tiers, including in-memory[8], burst-buffers[9], or parallel file systems. In this paper, we target in-memory DTL. The latter acts as a middle layer between the ensemble components (simulations/analyses) and the underlying DTL and is responsible for data handling. The simulation uses the DTL plugin to write out data abstracted into a *chunk*, which is the base data representation managed within the entire runtime. This abstraction allows the system to be adaptable to a variety of simulations and eases the burden of developing special-purpose code to pair with diverse simulation types. The chunk also defines a unique data type standard for the analysis kernels, though each of them may perform different computations. The DTL plugin performs data marshaling to support various DTL implementations. Specifically, the abstract chunk is serialized to a buffer of bytes, which is easy to manage for most DTL. The DTL plugin interfaces also hide the complexities of managing different I/O staging protocols in the DTL. To optimize the in situ data processing, coupled components in an ensemble member are synchronized as they progress concurrently over time. For example, in an ensemble of simulations, analysis steps can only execute upon completion of the current simulation step.

Experimental platform. Our execution platform is Cori[19], a Cray XC40 supercomputer located at the National Energy Research Scientific Computing Center (NERSC). Each compute node is equipped with two Intel Xeon E5-2698 v3 (16 cores each) sharing 128 GB of DRAM, which are connected through a Cray Aries dragonfly topology. To test the impact of co-locating the analyses and the simulation, we set the simulation to a predefined stride and choose the settings for the analysis that satisfy two conditions: (i) a simulation step takes longer than an analysis step so that the analysis does not slow down the simulation; (ii) the idle time in the analysis (waiting for simulations' chunks) is minimized, so that we maximize the time that the analyses and simulations are running at the same time. Section 3.4.1 provides more details about the approach. For our experiments, the two constraints are satisfied by the following resource allocations: every simulation runs on 16 physical cores of a computing node with a stride equal to $2,000$ and $30,000$ simulation steps, and each analysis uses 8 physical cores. We leverage DIMES[8] to deploy the in-memory staging area for the DTL. DIMES is an in situ implementation in which data is kept locally in the node memory on which the simulation is running and distributed over network to nodes upon request. We use TAU[20] to collect execution times, performance counters, and memory footprints. Measurements are averaged over five trials.

Workflow configurations. In this work, we experiment with workflow ensembles instantiated with different configurations (e.g., number of ensemble members, component placements) to study co-location behaviors. Table 2 shows the 7 configurations used in our experiments. These configurations include the number of ensemble members, number of computing nodes allocated for the entire workflow ensemble , and node indexes in the allocation on which each ensemble component is running. Every ensemble member is comprised of one simulation coupled with one analysis. C_f and C_c are two elementary configurations in which each configuration has a single ensemble member. C_f describes a co-location-free placement, i.e. the simulation and the analysis are located on two separate nodes. C_c co-locates the simulation and the analysis on a single compute node. The configurations for two ensemble members explore several co-location scenarios of ensemble components. In $C1.1$, the two analyses run on the same node and each simulation on a dedicated node; in $C1.2$, both simulations share a node and analyses run on dedicated nodes. In $C1.3$, the simulation and the analysis of the first ensemble member share the same node, while the other ensemble member has the simulation and the analysis running on two different nodes. In $C1.4$, the two simulations share a node and the two analyses share another node. Finally, $C1.5$ represents the setup where each simulation shares a node with its corresponding analysis.

Config-uration	Number of computing nodes	Number of ensemble members	Node indexes			
			Ensemble member 1		Ensemble member 2	
			Simulation 1	Analysis 1	Simulation 2	Analysis 2
C_f	2	1	n_0	n_1	-	-
C_c	1	1	n_0	n_0	-	-
C1.1	3	2	n_0	n_2	n_1	n_2
C1.2	3	2	n_0	n_1	n_0	n_2
C1.3	3	2	n_0	n_0	n_1	n_2
C1.4	2	2	n_0	n_1	n_0	n_1
C1.5	2	2	n_0	n_0	n_1	n_1

TABLE 2 Experimental scenarios configuration settings.

2.3 | Analyzing workflow ensemble co-location

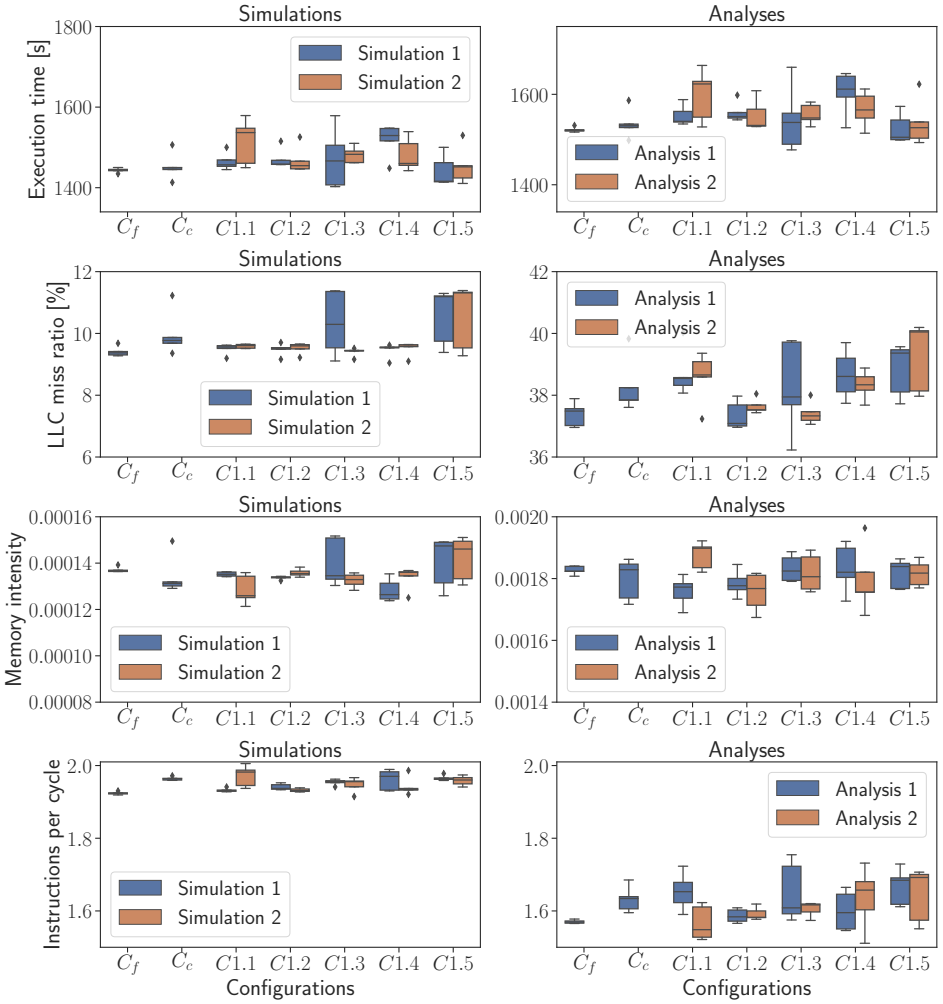

FIGURE 3 Metrics at ensemble component level.

Figures 3 to 5 show measurements obtained with the set of traditional metrics (Table 1) for the various configuration settings (Table 2). Higher LLC miss ratios in Figure 3 (compared to co-location-free configuration C_f) capture the cache misses in C_c, and $C1.1$ to $C1.5$ due to resource contention from the co-located ensemble components. In our application, analyses are more memory-intensive than the simulations, thus co-locations of the analyses, i.e. $C1.1$ and $C1.4$, result in higher cache misses than the co-location of the simulations, i.e. $C1.2$. The co-location of heterogeneous tasks (the simulation and the analysis) lead to higher miss rates in $C1.3$ and $C1.5$ compared to $C1.1$, $C1.2$, and $C1.4$. That said, $C1.5$ yields the shortest member makespan among all configurations (Figures 4 and 5). We argue that co-locating coupled components within an ensemble member leads

to execution efficiency despite the elevated degree of LLC interference. However, only simulation and analyses that exchange data should be co-located.

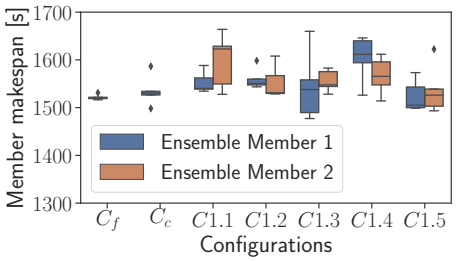

FIGURE 4 Ensemble member makespan.

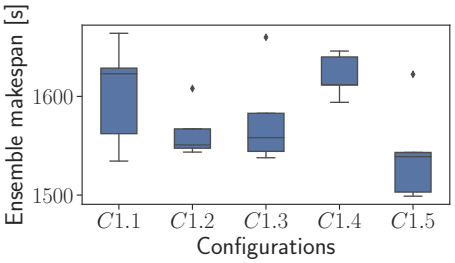

FIGURE 5 Workflow ensemble makespan.

The overall conclusion is that evaluating each set of metrics exclusively does not guarantee a thorough understanding of the workflow ensemble performance. Metrics at the component level yield insights into the characteristics of individual components, but fail to capture the overall workflow ensemble behavior. For example, in our case, analyses are more memory-intensive than simulations, which leads to increased cache miss ratio or higher memory interference. As a result, resource contention may arise due to co-located analyses, thereby not only leading to increased execution time of these components, but also increased ensemble member makespan (recall the simulation and analyses execute synchronously). Consequently, the overall workflow ensemble makespan may be harmed due to slow ensemble members. Therefore, in order to identify stragglers among the members one would need to diligently inspect and relate the independent measurements to draw conclusions of the workflow ensemble performance. We argue then that there is a need to develop a method that captures the performance within a workflow ensemble at multiple levels of granularity. To this end, in the next section, we present an efficiency metric that indicates effective computation during the execution of an ensemble member. We then consolidate measurements collected at the ensemble member level into an indicator of overall workflow ensemble efficiency.

3 | EFFICIENCY MODEL

To assess the performance of the workflow ensembles, we first address the demand of execution characterization at the level of ensemble members. In this section, we present an in situ execution model for a single ensemble member. Based on this model, we propose an indicator to estimate the computational efficiency for an ensemble member. We expanded the single simulation/single analysis model presented in[21,22] to include multiple analysis components coupled to a single simulation (Figure 1). We then leverage this efficiency indicator as one of the prerequisites to synthesize the performance of workflow ensembles in Section 4.

3.1 | Application Model

In our model, every simulation step is divided into three fine-grained stages: (i) a simulation stage S, (ii) an idle stage I^S, and (iii) a writing stage W in order, i.e. S occurs before I^S, I^S happens before W. The simulation performs the computation during S, waits for the time when data are ready to stage in I^S, and then sends data to the analysis during W. Similarly, every analysis step is comprised of: (i) a reading stage R, (ii) an analyzing stage A, and (iii) an idle stage I^A, executed in that order. The analysis reads data sent by the simulation in R, performs certain analyses during A, and then waits until the next chunk of data is available for processing during I^A. These fine-grained stages can be organized into three sub-groups: (i) computational stages (S, A), (ii) I/O stages (W, R), and (iii) idle stages (I^S, I^A).

The synchronous communication pattern discussed in Section 2 enforces the coordination among I/O stages such that W_i of step i occurs before R_i, and R_i happens before W_{i+1} of the next iteration (Figure 6) so that the simulation does not overwrite data, which have not been read yet (i.e., we assume no buffering of the simulation output, in conformity with[21]). Thanks to the iterative relationship between simulations and analyses, their executions, after a few warm-up steps, reach a steady-state where each stage has a similar execution time as measure over many steps. As a result, rather than considering a particular step i for a given stage (e.g., W_i), we use a *star* symbol to denote steady-state stages. Then, S_*, I^S_*, W_*, R_*, A_*, and I^A_* denote the steady-state stages of S, I^S, W, R, A, and I^A respectively.

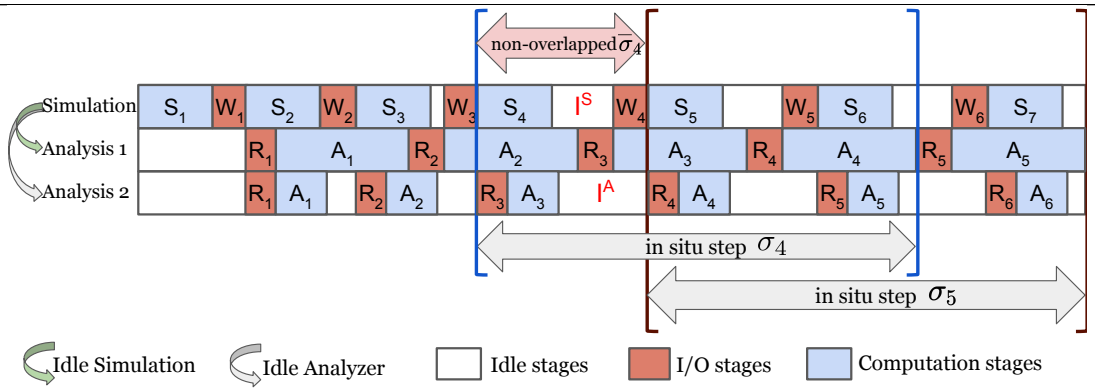

FIGURE 6 Fine-grained execution steps for one ensemble member.

3.2 | In Situ Step

A given ensemble member is composed of a single simulation Sim coupled with K analyses $Ana^1, Ana^2, \ldots, Ana^K$. An in situ step is defined as the duration between the beginning of the stage S in the simulation and the end of the stage I^A that finishes last among the K analyses. We characterize the execution of a coupled simulation-analysis into two scenarios (Figure 6): (i) *Idle Simulation* – a given analysis step runs longer than the corresponding simulation step; (ii) *Idle Analyzer* – a given analysis step runs faster than the associated simulation step. In Idle Simulation, the simulation step waits for the completion of the analysis step. In contrast, in Idle Analyzer the analysis step waits for data available from the corresponding simulation step. For example, in Figure 6, the coupling of the simulation and the analysis 1 falls into the Idle Simulation scenario, while the simulation and the analysis 2 are paired under the Idle Analyzer scenario.

An ensemble member with one simulation and K analyses has K different couplings $\{(Sim, Ana^1), \ldots, (Sim, Ana^K)\}$ shortened in this work as (Sim, Ana^i) with $1 \leq i \leq K$. (Each of these couplings can be categorized as either Idle Simulation or Idle Analyzer scenarios.) Note that multiple in situ steps may overlap due to concurrent executions. Thus, computing the makespan of an ensemble member should also account for this behavior – by simply expressing the makespan as the aggregation of in situ steps execution times, its value is likely to be overestimated. As a result, we define an "actual" in situ step as the non-overlapped segment $\overline{\sigma}_*$ (Figure 6).

Intuitively, the non-overlapped segment $\overline{\sigma}_*$ of a given in situ step is the section between two consecutive simulation stages S (recall an in situ step starts with the stage S). There are two possible scenarios: (i) the simulation and the write stage run longer (Idle Analyzer scenario), then the non-overlapped segment is equals to $S_* + W_*$; or (ii) one of the K analysis, Ana^i, has the longest execution time (Idle Simulation scenario) then, the non-overlapped step is equals to $R^i_* + A^i_*$. Hence,

$$\overline{\sigma}_* = \max(S_* + W_*, R^1_* + A^1_*, \ldots, R^K_* + A^K_*). \tag{1}$$

Given the non-overlapped segment of in situ steps, we compute the execution time of one ensemble member (also known as the makespan) as $\text{MAKESPAN} = n_{steps} \times \overline{\sigma}_*$, where n_{steps} is the total number of in situ steps. As MAKESPAN is first-order approximation, we can omit the remaining portion of the last in situ step. This approximation holds when n_{steps} is large enough[21].

3.3 | Computational Efficiency

To characterize the execution of an ensemble member, in this section, we propose an *indicator* to capture the efficiency of the execution of an ensemble member from a computational standpoint, where we want to minimize the idle time while increasing resource usage. To compute the idle time per in situ step, we use Equation (1) to derive the duration of the idle stage on the simulation component: $I^S_* = \overline{\sigma}_* - (S_* + W_*)$ and, the duration of the idle stage for the analysis i as $I^{A_i}_* = \overline{\sigma}_* - (A^i_* + R^i_*)$. For each coupling (Sim, Ana^i), the portion of effective computation, i.e. not sitting idle, of an actual in situ step is defined as $\overline{\sigma}_* - (I^S_* + I^{A_i}_*)$. Since the computational efficiency of an ensemble member depends on the amount of time the ensemble components are idle, we compute a computational efficiency E as the average time of effective computation over the actual in situ step of K couplings in the ensemble member:

$$E = \frac{1}{K}\sum_{i=1}^{K}\left(1 - \frac{I^S_* + I^{A_i}_*}{\overline{\sigma}_*}\right) = \frac{S_* + W_*}{\overline{\sigma}_*} + \frac{\sum_{i=1}^{K}(A^i_* + R^i_*)}{K\,\overline{\sigma}_*} - 1. \tag{2}$$

This indicator is derived from $\overline{\sigma}_*$, hence maximizing E implies minimizing the idle time and thereby the makespan.

3.4 | Discussion

3.4.1 | Choice of Settings

In this section, we use our efficiency model to substantiate the choice of settings (i.e., number of cores) used to run the experiments shown in Section 2.2. Recall that for that set of experiments, we consider one MD simulation coupled with one in situ analysis. The parameter space is intractable as we can vary the number of cores per component, their respective placements, and the stride of the simulation. Thus, an exhaustive search is out of reach. However, we can define a heuristic that seeks for parameters that minimize the makespan and maximize the computational efficiency of an ensemble member. In this context, we make the following assumptions:

- The simulation settings are considered as an input of the problem and are provided by the user. In most cases, scientists have a rough estimate of the best settings for their simulations, but not for the analyses.
- Although our theoretical framework supports coupling to different types of analyses simultaneously, we limit our experiments to only identical analyses.

We first consider the scenario without co-location, and we argue that settings provisioned to the simulation and the analysis within that context act as a baseline when contrasting to scenarios with co-location. In this experiment, to ensure that there is no co-location, we consider a simple coupling of a single MD simulation coupled with one analysis executed on one dedicated compute node. Based on our first assumption, we arbitrarily set the settings of the simulation as follows: 8 cores and a stride of 2000. We then vary the number of cores allocated to the analysis to determine for which number of cores the makespan is minimized and the computational efficiency E is maximized in that configuration (recall that our execution platform has compute nodes embedding 32 cores).

We know that minimizing the makespan is equivalent to minimizing $\overline{\sigma}_*$ (see Section 3.2). Thus, given an ensemble member with a certain simulation and a predefined configuration coupled with in situ analyses, in order to minimize the makespan, we need to assign a number of cores to the analysis such that: $R^i_* + A^i_* \leq S_* + W_*, \forall i \in \{1, 2, \ldots, K\}$. This inequality implies that each of the K coupling (Sim, Ana^i) falls into the Idle Analyzer scenario so that the analysis steps are hidden by the simulation steps to not slow down the makespan. Figure 7 shows the impact, when the number of cores assigned to the analysis ranges from 1 to 32, on the in situ step $\overline{\sigma}_*$, the simulation component $S_* + W_*$, the analysis component $R_* + A_*$, and the computational efficiency E. The analysis step when using 1 and 2 cores takes longer than the simulation step, i.e. $R_* + A_* > S_* + W_*$, thus $\overline{\sigma}_* = R_* + A_*$. The inequality is satisfied once the analysis uses between 4 and 32 cores, which minimizes $\overline{\sigma}_* = S_* + W_*$, thereby minimizing the member makespan. Among executions whose makespan are minimized, we optimize the computation efficiency by selecting the configuration that leads to $max(E)$. Hence, we decide to assign 4 cores to each analysis, which results in the highest computational efficiency, i.e. the smallest amount of idle time.

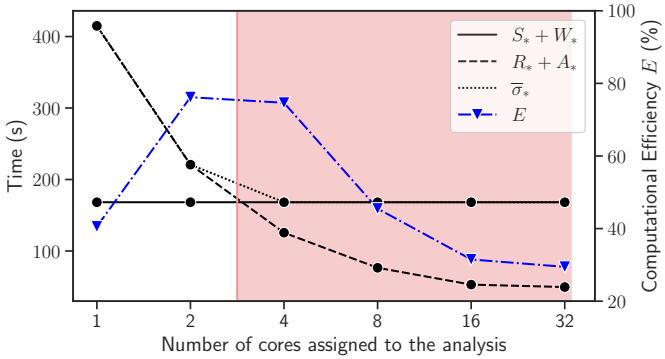

FIGURE 7 Execution time of the in situ step and computational efficiency when varying the number of cores assigned to the analysis with a fixed simulation setting.

3.4.2 | Impact of Co-location

In this section, we estimate the impact of co-locating ensemble components within a workflow ensemble by conducting executions of workflow ensembles with two members on three configurations described in Table 3 (each configuration uses two compute nodes). $C_{colocated}$ co-locates the simulation and the analyses of an ensemble member on the same compute node. $C_{dedicated}$ co-locates two simulations on the same node while all the analyses are co-located on the second dedicated node. Finally, in C_{hybrid} the analyses are placed on the node on which the simulation of the other ensemble member is running. A compute node has 32 cores and, for all configurations, every simulation is assigned 8 cores and every analysis 4 cores. Recall from Section 3.4.1 this setting minimizes idle time when there is no co-location between simulation and analyses for ensembles with one simulation coupled with one analysis. In this experiment, we increase the number of analyses per ensemble member to observe the impact of co-location among ensemble components.

Configuration	Number of computing nodes (N)	Number of ensemble members	Node indexes			
			Ensemble member 1		Ensemble member 2	
			Simulation 1	Analyses 1.x	Simulation 2	Analyses 2.x
$C_{colocated}$	2	2	n_0	$n_0, \ldots, n_0$	n_1	$n_1, \ldots, n_1$
$C_{dedicated}$	2	2	n_0	$n_1, \ldots, n_1$	n_0	$n_1, \ldots, n_1$
C_{hybrid}	2	2	n_0	$n_1, \ldots, n_1$	n_1	$n_0, \ldots, n_0$

TABLE 3 Experimental configurations with two ensemble members, each ensemble member has two analyses per simulation.

Figure 8 shows the computational efficiency corresponding to the three configurations when the number of analyses per ensemble member ranges from 1 to 4, each data point is the result of 5 trials. Overall, we observe higher efficiencies in $C_{colocated}$ and C_{hybrid} than $C_{dedicated}$ at small number of analyses per ensemble member (1 and 2). Several studies have demonstrated the benefits of co-locating compute-intensive with memory-intensive applications[23,24]. Our findings confirm the benefits of co-locating heterogeneous applications, i.e. compute-intensive simulation and memory-intensive analyses, that triggers less resource contention (recall from Section 2.3, the analysis is memory-intensive while the simulation is compute-intensive).

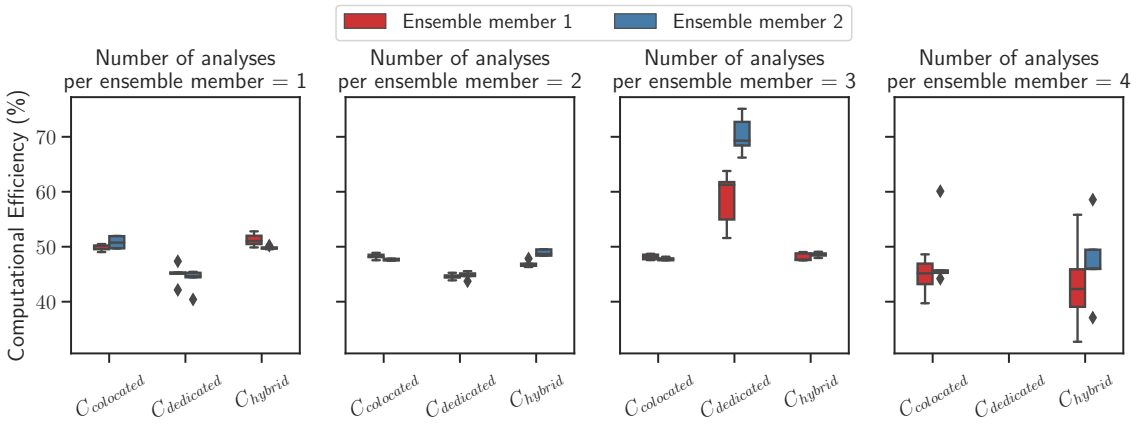

FIGURE 8 Computational efficiency when varying the number of analyses per ensemble member. Note that the missing values in the $C_{dedicated}$ configuration when running with 4 analyses per ensemble member is due to out of memory errors on the node.

Executions with 3 analyses per ensemble member result on unexpectedly high efficiency values for $C_{dedicated}$, which is due to the dramatically slowing down of the analyses once co-locating with large enough number of memory-intensive analyses (6 analyses in total for 2 ensemble members) on a single node. This increase in efficiency for $C_{dedicated}$ indicates a small amount of time sitting idle during the execution, however, when compared to the other two configurations, it also implies fewer idle resources remaining to accommodate more analyses. The good efficiency demonstrated by $C_{dedicated}$ when running with 3 analyses is due to co-locating three memory-intensive applications together, which leads to performance degradation due to competitions for the shared resources, thus the analysis side is slowing down and getting closer to the simulation execution time, hence leading

to an overall smaller idle time and efficiency, that can be seen as a “negative improvement". Inappropriate co-location strategies can lead to poor performance of in situ workflow ensembles, but, even more dramatic, looking at efficiency solely can lead to poor choices of placement strategies. These findings require broader considerations, beside computational efficiency, when designing workflow ensemble placement strategies to avoid such negative improvements which are misleading.

Computational efficiency is not sufficient without considering the resource specification of a given configuration that reflects how efficiently the underlying resources are utilized. Without considering resource aspects, i.e., number of cores assigned to each ensemble components, total number of nodes, or the placement of ensemble components, the executions of two workflow ensembles are not comparable. As demonstrated in Figure 8, different ensemble members exhibit different computational efficiencies, thus synthesizing the overall performance of workflow ensembles with many ensemble members requires summarizing a large number of efficiency values, which is not straightforward. We acknowledge these limitations and, in the next section, we introduce the concept of *performance indicators* that aims to address these limitations.

4 | PERFORMANCE INDICATORS

Traditionally, scientists running on HPC machines want to optimize applications performance while using as few resources as possible. When considering multiple concurrent components like workflow ensembles, defining the notion of resource usage and its perimeter is already challenging (e.g., each ensemble member can use different numbers of cores, ensemble components can have various mapping onto allocated resources). In addition, since ensemble members are executed simultaneously, one needs to consider their local resource usages and performance but also define a method to aggregate local knowledge into a coherent global analysis. As detailed in Section 3.4.2, efficiency by itself is not sufficient to describe such complex concurrent executions and does not consider the underlying resource usage.

4.1 | Framework Definition

In this section, we define a framework, denoted as *performance indicators*, that provides us with a method to aggregate resource usage of different members within a workflow ensemble. We augment the notion of efficiency previously described with resource context under the form of a multi-stage framework that aims to capture the efficiency of every ensemble member under multiple resource constraints. Each stage of the framework adds a layer of information to the performance indicator that characterizes a certain resource feature, such as number of resources used and resource mapping.

More formally, given a workflow ensemble with $1 \leq i \leq N$ ensemble members $\{EM_1, \dots, EM_N\}$, let E_i be the computational efficiency of EM_i (as defined in Equation (2)). We first define R_i, a given resource constraint affecting member EM_i, for example we could define R_i as the number of cores allocated to EM_i. Then, we can formulate the problem of optimizing the global efficiency of a workflow ensemble under possibly several resource constraints R_i as follows:

$$\begin{aligned} &\text{maximize} \quad E_i \quad , \forall i = 1, \dots, N \\ &\text{subject to} \quad \text{minimize/maximize } R_i. \end{aligned} \tag{3}$$

The idea is to maximize efficiency of all ensemble members but under multiple predefined constraints. These constraints can be arbitrarily chosen by users (e.g., number of cores, network links capacity). For convenience, the efficiency E_i of each ensemble member EM_i and the constraint R_i are combined and rewritten as a performance indicator P_i, which can be seen as a function of E_i. More precisely, $P_i = E_i \times R_i$ if the constraint is to maximize R_i, otherwise $P_i = E_i / R_i$ if the constraint is to minimize R_i. For example, let c_i be the number of cores allocated to EM_i to be minimized, then to design a performance indicator considering the number of cores, we would define $R_i = c_i$ and $P_i = E_i / c_i$ (recall we maximize efficiency while minimizing the resource indicator, so we have to divide in that situation). With these performance indicators, Equation (3) can be simply rewritten as:

$$\text{maximize} \quad P_i \quad , \forall i = 1, \dots, N. \tag{4}$$

Now that we have a framework to evaluate executions of workflow ensembles, we need a method to aggregate information from each performance indicator into one coherent measure. To synthesize the performance of a workflow ensemble with potentially many members, we propose a method that accumulates performance indicators P_i of every ensemble member using an objective function F (defined in Section 4.6). Therefore, the problem stated in Equation (4) can be transformed into its final form:

$$\text{maximize} \quad F(P_1, \dots, P_N). \tag{5}$$

The goal of this whole process is to provide a methodology to assess the impact of each layer of resource information R_i and obtain an overall indicator that can characterize the performance of the entire workflow ensemble. We discuss the procedure for calculating performance indicators and generating the objective function to aggregate them in the following sections. First, we define a set of notations (Table 4) used to define the indicators. Then, we present three resource indicators $R^{\mathrm{U}}, R^{\mathrm{A}}, R^{\mathrm{P}}$ corresponding, respectively, to resource usage (U), resource allocation (A), and resource provisioning (P).

4.2 | Notations

The ensemble member EM_i contains a simulation Sim_i coupled with K_i analyses, $Ana_i^1, \ldots, Ana_i^{K_i}$, thus EM_i has K_i couplings (Sim_i, Ana_i^j), where $j \in \{1, \ldots, K_i\}$. Let cs_i be the number of cores used by Sim_i, these cores belong to nodes whose indexes are listed in set s_i. Similarly, the analysis Ana_i^j uses ca_i^j cores of nodes whose indexes are defined in set a_i^j. For example, in Table 2, C1.1 has $s_1 = \{0\}, a_1^1 = \{2\}, s_2 = \{1\}, a_2^1 = \{2\}$. Let c_i denote the total number of cores assigned to all ensemble components, i.e. simulation Sim_i and K_i analyses Ana_i^j, in a given ensemble member EM_i. We have $c_i = cs_i + \sum_{j=1}^{K_i} ca_i^j$. Let d_i be the number of computing nodes allocated to the ensemble member EM_i. Then, the number of compute nodes d_i allocated to the ensemble member EM_i is calculated by $d_i = \left| s_i \cup \bigcup_{j=1}^{K_i} a_i^j \right|$. If the simulation and some analyses share compute nodes, we have $d_i \leq |s_i| + \sum_{j=1}^{K_i} |a_i^j|$. (Note that this inequality becomes an equality if each component runs on dedicated nodes.) Let M be the total number of computing nodes used by the entire workflow of N ensemble members. Similarly, we have $M \leq \sum_{i=1}^{N} d_i$. In the absence of resource sharing (i.e, each ensemble member runs on dedicated nodes), we have $M = \sum_{i=1}^{N} d_i$.

Notation	Description
	Workflow Ensemble
N	Number of ensemble members
M	Number of compute nodes used by the workflow ensemble
	Ensemble Member
EM_i	Ensemble member i
R_i	Resource constraint affecting EM_i
P_i	Performance indicator of EM_i
K_i	Number of couplings in EM_i
c_i	Total number of cores used by components of EM_i
d_i	Number of nodes allocated to EM_i
	Ensemble Component
Sim_i	Simulation of EM_i (one simulation per member)
Ana_i^j	Analysis j of EM_i (K_i analysis for each EM_i)
cs_i	Number of cores used by Sim_i of EM_i
ca_i^j	Number of cores used by Ana_i^j from EM_i
s_i	Set of node indexes on which Sim_i from EM_i is executed
a_i^j	Set of node indexes on which Ana_i^j from EM_i is executed

TABLE 4 Notations.

4.3 | Member Resource Usage (U)

The first performance indicator P_i^{U} considers underlying computing units, i.e. cores, to model the efficiency of an ensemble member in terms of resource usage. Our goal is to build an indicator that can compare different executions of workflow ensembles using different numbers of resources (e.g., number of cores). Precisely, P_i^{U} maximizes computational efficiency E_i of an ensemble member EM_i such that the total number of cores c_i used by EM_i is minimized. We then define the resource usage indicator $R_i^{\mathrm{U}} = c_i$. To minimize R_i^{U}, P_i^{U} is computed as follows:

$$P_i^{\mathrm{U}} = \frac{E_i}{R_i^{\mathrm{U}}} = \frac{E_i}{c_i} \tag{6}$$

P_i^{U} represents the smallest unit of efficiency in terms of single core usage. Recall that maximizing E_i is equivalent to minimizing the idle time and the makespan (Section 3.3). High values of P_i^{U} indicate that a large portion of the execution on assigned resources is spent on computing (in contrast to idling), thus the ensemble member makespan is reduced.

4.4 | Member Resource Allocation (A)

Since an ensemble member can have concurrent execution of multiple components, the component can be co-located on the same node or distributed across nodes. Finding an optimal placement among the numerous placement configurations is challenging. Therefore, we propose the second stage P_i^{A} to quantify the level of data locality of a certain placement.

Lets consider the coupling (Sim_i, Ana_i^j) part of the ensemble member EM_i, then Sim_i is co-located with Ana_i^j if and only if $|s_i| = |s_i \cup a_i^j|$. Otherwise, if $|s_i| < |s_i \cup a_i^j|$, then they are assigned to different nodes. Based on this observation, we define a *placement indicator* obtained from the ratio $0 < \frac{|s_i|}{|s_i \cup a_i^j|} \leq 1$ to represent a placement of a workflow ensemble. Let CP_i be the placement indicator for the ensemble member EM_i:

$$CP_i = \frac{1}{K_i}\left(\frac{|s_i|}{|s_i \cup a_i^1|} + \cdots + \frac{|s_i|}{|s_i \cup a_i^{K_i}|}\right) = \frac{|s_i|}{K_i}\sum_{j=1}^{K_i}\frac{1}{|s_i \cup a_i^j|}. \tag{7}$$

Intuitively, CP_i describes the placement of EM_i. It decreases with the number of computing nodes used for a given coupling. $CP_i = 1$ indicates that the EM_i components are all co-located, and a CP_i value near 0 indicates that more dedicated resources are used and that the components of EM_i are distributed across them. Maximizing the placement indicator for each ensemble member results in prioritizing placements that minimize the number of computing resources used by that ensemble member. As a result, the placement indicator not only reflects placement characteristics but also the number of resources used at the ensemble member level.

To evaluate the efficiency of a placement (i.e., a mapping between ensemble members and available resources), we include the proposed placement indicator as the resource indicator R^{A} in the next stage of the performance indicator. Specifically, we multiply the first stage of our performance indicator by $R_i^{\mathrm{A}} = CP_i$ as follows:

$$P_i^{\mathrm{A}} = E_i \times R_i^{\mathrm{A}} = E_i \times CP_i = E_i\frac{|s_i|}{K_i}\sum_{j=1}^{K_i}\frac{1}{|s_i \cup a_i^j|}. \tag{8}$$

Based to the insight derived from the placement indicator, maximizing the performance indicator at this stage favors the resource configuration that occupies a small number of compute nodes while maximizing the effectiveness of the execution.

4.5 | Ensemble Resource Provisioning (P)

Finally, by just considering the execution features at the ensemble member level might not be sufficient to capture the overall performance of the entire workflow ensemble. To that end, we extend the performance indicator with the number of resources provisioned for the entire workflow ensemble, i.e. the number of computing nodes the workflow ensemble resides on. When comparing two executions using different number of computing nodes, the run using the smaller number of nodes should yield better efficiency in two settings with the same performance. Therefore, to obtain the last stage P_i^{P}, we weigh the efficiency indicator by $R_i^{\mathrm{P}} = M$, where M is the total number of compute nodes, so that the number of compute nodes provisioned for the entire workflow ensemble is minimized while P_i^{P} is maximized:

$$P_i^{\mathrm{P}} = \frac{E_i}{R_i^{\mathrm{P}}} = \frac{E_i}{M}. \tag{9}$$

Finally, depending on the resource aspects of interest, the performance indicator P_i can either represent a single-stage indicator P_i^{U}, P_i^{A}, P_i^{P}, or a multi-stage indicator $P_i^{\mathrm{U,A}}$, $P_i^{\mathrm{U,P}}$, $P_i^{\mathrm{A,P}}$, $P_i^{\mathrm{U,A,P}}$. For example, $P_i^{\mathrm{U,A,P}} = \frac{E_i \times R_i^{\mathrm{A}}}{R_i^{\mathrm{U}} \times R_i^{\mathrm{P}}}$.

4.6 | Objective Function

In this section, we propose a method for aggregating indicator values from individual ensemble members into a global indicator at the workflow ensemble level. In order to compute a global indicator, we synthesize performance indicators of every ensemble member. A simple approach could consider the average values for all P_i. However, the large variation between these values may lead to an inaccurate assessment of the overall performance. To minimize the variability in performance among ensemble

members, we consider the mean performance $\overline{P}$ from which we subtract the standard deviation:

$$F(P_i) = \overline{P} - \sqrt{\frac{1}{N}\sum_{i=1}^{N}(P_i - \overline{P})^2} \quad \text{where} \quad \overline{P} = \frac{1}{N}\sum_{i=1}^{N} P_i. \tag{10}$$

The intuition behind Equation (10) is to favor workflow ensemble's configurations with good makespan, i.e. configurations with low variability between workflow ensemble members (recall that the makespan of a workflow ensemble is defined as the maximum completion time among its members). The goal of an efficient configuration, as defined in this work, is to maximize the objective function F. The higher the value of the objective function, the better the performance of the entire workflow regarding efficiency, makespan, resource usage, and component placement.

5 | EXPERIMENTAL EVALUATION

In this section, we evaluate the ability of the proposed performance indicators to characterize the execution performance of workflow ensembles. We extend our previous experimental configuration settings (Section 2.2) with scenarios in which multiple analyses are coupled with the simulation.

5.1 | Configuration Exploration

Configuration	Number of computing nodes (N)	Number of ensemble members	Node indexes					
			Ensemble member 1			Ensemble member 2		
			Simulation 1	Analysis 1.1	Analysis 1.2	Simulation 2	Analysis 2.1	Analysis 2.2
C2.1	3	2	n_0	n_2	n_2	n_1	n_2	n_2
C2.2	3	2	n_0	n_1	n_1	n_0	n_2	n_2
C2.3	3	2	n_0	n_1	n_2	n_0	n_1	n_2
C2.4	3	2	n_0	n_0	n_2	n_1	n_1	n_2
C2.5	3	2	n_0	n_1	n_2	n_1	n_0	n_2
C2.6	2	2	n_0	n_1	n_1	n_0	n_1	n_1
C2.7	2	2	n_0	n_0	n_1	n_1	n_0	n_1
C2.8	2	2	n_0	n_0	n_0	n_1	n_1	n_1

TABLE 5 Experimental configurations with two ensemble members, each ensemble member has two analyses per simulation.

Workflow ensemble configurations. In this work, we apply our multi-stage performance indicators to two sets of configurations, each of these sets specifies the number of ensemble members and the node assignment for each ensemble component. In this paper, we consider only workflow ensembles comprised of 2 ensemble members. The first set of configurations includes $C1.1$ to $C1.5$ (Table 2). For every configuration in this set, each ensemble member is a single coupling of a simulation and an in situ analysis. The second set consists of configurations ranging from $C2.1$ to $C2.8$ (Table 5). For configurations in this set, the simulation of each ensemble member is coupled with two analyses. For each configuration in both sets, every simulation runs on 16 cores while every analysis is assigned 8 cores, which is identified by following the similar procedure described in Section 3.4.1 to minimize idle time occurred in the coupling between them. With this setting, configurations of the second set leverage all cores of each compute node, thus saturating the computing resources (recall that each compute node has 32 cores). Since we propose a multi-stage method for evaluating the performance of an ensemble member as well as the entire workflow ensemble, we examine the impact and the order of each stage on the quality of the performance indicator P_i by accumulating in the objective function $F(P_i)$ as the performance of the entire workflow ensemble. To this end, we explore two feasible paths that can be followed to concatenate performance indicator stages: (1) $P_i^{\mathrm{U}} \rightarrow P_i^{\mathrm{U,P}} \rightarrow P_i^{\mathrm{U,P,A}}$; or (2) $P_i^{\mathrm{U}} \rightarrow P_i^{\mathrm{U,A}} \rightarrow P_i^{\mathrm{U,A,P}}$. For path (1), $P_i^{\mathrm{U,P}} = P_i^{\mathrm{U}}/M$, where M is the total number of nodes used by the workflow ensemble (see Table 4) and $P_i^{\mathrm{U,P,A}} = P_i^{\mathrm{U,P}} \times CP_i$, where CP_i is the placement indicator defined in Section 4.4. Note that $P_i^{\mathrm{U,P,A}} = P_i^{\mathrm{U,A,P}}$. Specifically, we observe changes in $F(P_i)$ when adding a new stage (i.e., resource usage U, resource provisioning P, resource allocation A) to the performance indicator P_i, which can be either P_i^{U}, $P_i^{\mathrm{U,P}}$, $P_i^{\mathrm{U,A}}$, $P_i^{\mathrm{U,P,A}}$, or $P_i^{\mathrm{U,A,P}}$.

Results. Figure 9 demonstrates the results of the objective performance function at each of the multiple stages of P_i over different configurations in the first set. After the initial stage of P_i^{U} (Figure 9 left), a new layer is added, either P in the middle top figure

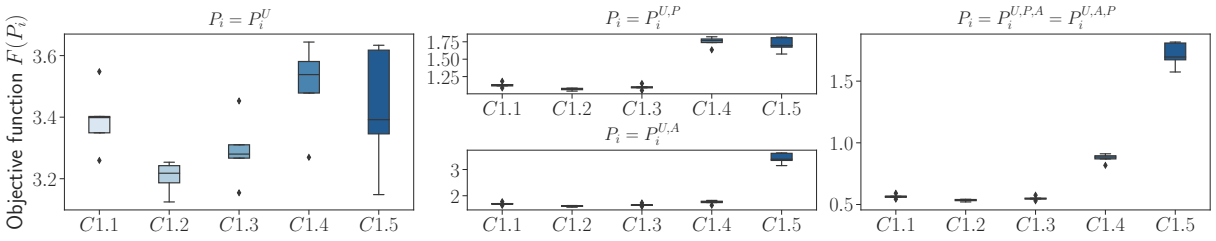

FIGURE 9 $F(P_i)$ on different P_i orders over configurations which have one analysis per simulation (the higher the better).

or A on the middle bottom to form the next stage. On the contrary, $P_i^{\text{U,A}}$, $P_i^{\text{U,P}}$ is not able to differentiate the performance of $C1.4$ from $C1.5$ as these two configurations both use 2 compute nodes. Recall that in $C1.4$ the two simulations share a node while the two analyses share another node. As shown in Figures 3 and 4, $C1.4$ does not lead to small member makespan due to the contention of co-location between two analyses. In $P_i^{\text{U,A,P}}$, the performance of $C1.4$ is degraded to lower than $C1.5$, but higher than $C1.1$, $C1.2$, and $C1.3$. Finally, our performance indicator confirms that $C1.5$ is the best choice, as demonstrated by traditional metrics in Figures 4 and 5 that $C1.5$ has the smallest makespan. $C1.5$ outperforms other configurations, which also validates the common intuition associated with in situ processing that simulations and analyses must be co-located when possible. Since the in-memory staging mechanism in this work is implemented by DIMES[8] (data resides on the memory of the simulation node), co-locating the analysis with the simulation can benefit from data locality – the time for staging data is significantly shortened.

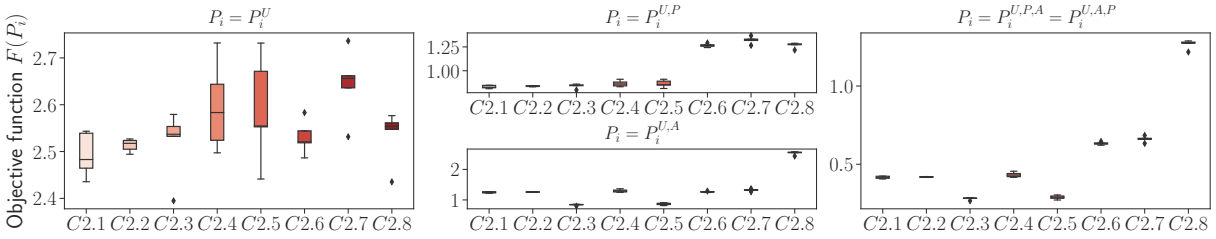

FIGURE 10 $F(P_i)$ on different P_i orders over configurations which have two analyses per simulation (the higher the better).

By opposition to the first set of configurations, for the second set, we do not show the results of traditional metrics (described in Table 1) due to the lack of space. However, experimental results of these metrics when using the second set of configurations are not as straightforward as the first on inferring from the metrics monitored which configuration is the best. The increased number of analyses involved in an ensemble member complicates the performance evaluation using traditional metrics. Utilizing the whole cores of compute nodes in several configurations, e.g. $C2.6, C2.7, C2.8$, likely saturates the resources, which makes it difficulty to compare them with other configurations where compute nodes are not entirely occupied by ensemble components. This scenario motivates the need for a performance indicator able to elect the best potential configuration in terms of efficiency of the workflow ensemble. Figure 10 shows the values taken by the objective function when instantiated with different configurations in the second set. In this case, $P_i^{\text{U,P}}$ separates the set of configurations in two groups defined by the number of compute nodes used by the workflow ensemble ($C2.6$, $C2.7$ and $C2.8$ uses 2 nodes when the other configurations use 3 nodes). Then, $P_i^{\text{U,P,A}}$ keeps this distinction but in addition indicates that configuration $C2.8$ should return better performance than the others. On the other hand, when adding layer A, we first isolate $C2.8$ from the other configurations, and further differentiate $C2.6, C2.7$ from $C2.1, C2.2, C2.4$ at the last stage. Note that, similarly to conclusions reached in the previous setup, the chosen configuration $C2.8$ is also the optimal configuration in terms of co-location (i.e, simulation is collocated with its analyses) which again confirms the benefits of co-locating coupled components of an ensemble member.

5.2 | Increased Number of Ensemble Members

Workflow ensemble configurations. In this section, we refine the three configurations described in Table 3 to scale up the number of ensemble members. Our goal is to increase the load on compute nodes' resources and increase network communications. We pile up 4 ensemble members on 2 nodes and explore different component placements (see Table 6). Specifically, $C_{colocated}$ co-locates ensemble components of every two ensemble members on the same node to guarantee data locality among ensemble

Configuration	Node indexes							
	Ensemble member 1		Ensemble member 2		Ensemble member 3		Ensemble member 4	
	Simulation 1	Analysis 1.1	Simulation 2	Analysis 2.1	Simulation 3	Analysis 3.1	Simulation 4	Analysis 4.1
$C_{colocated}$	n_0	n_0	n_0	n_0	n_1	n_1	n_1	n_1
$C_{dedicated}$	n_0	n_1	n_0	n_1	n_0	n_1	n_0	n_1
C_{hybrid}	n_0	n_1	n_0	n_1	n_1	n_0	n_1	n_0

TABLE 6 Experimental configurations for the first 4 ensemble members allocated on 2 compute nodes, each ensemble member has one simulation and one analysis. To increase the number of ensemble members, these settings can be replicated with a higher number of nodes (e.g., 8 ensemble members on 4 compute nodes, 16 ensemble members on 8 compute nodes).

components of an ensemble member. On the other hand, $C_{dedicated}$ co-locates simulations of every four ensemble members on a single node while the corresponding analyses are placed on another dedicated node. With C_{hybrid} configuration, ensemble components from a two-ensemble member pair are interchangeably placed together, i.e. the analyses of the other two ensemble members are co-located with the simulations of another two ensemble members. To accommodate sufficient cores for 4 ensemble members on 2 compute nodes, we couple a simulation with one analysis per ensemble member, in which the simulation used 8 cores and the analysis 4 cores. To increase the number of ensemble members per workflow ensemble, we replicate the placement described for 4 ensemble members. In this experiment, the number of ensemble members varies between 4 and 32.

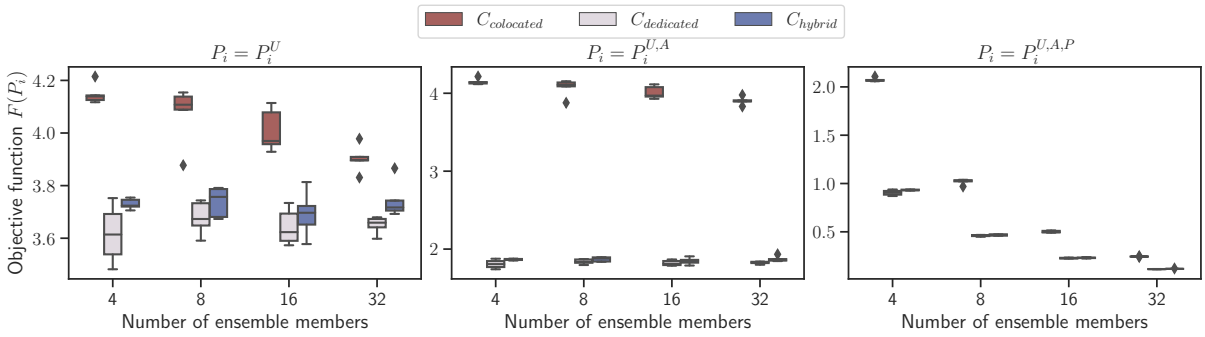

FIGURE 11 $F(P_i)$ on different P_i orders over configurations which have one analysis per simulation (the higher the better). Each configuration is measured over 5 trials.

Results. Figure 11 shows the values of the objective function for each performance indicator P_i^{U}, $P_i^{\mathrm{U,A}}$, $P_i^{\mathrm{U,A,P}}$. With $P_i = P_i^{\mathrm{U}}$, since the number of cores used by every ensemble member is identical among configurations, P_i^{U} indirectly reflects the computational efficiency of each configuration. Recall from Section 3.4.2 that the computational efficiency of $C_{colocated}$ and C_{hybrid} are approximately comparable once there is only one simulation and one analysis co-located on a single node. We note that $C_{colocated}$ surpasses C_{hybrid} when a greater number of ensemble components competes for a certain amount of resources. This observation highlights the significance of data locality for allocating ensemble components of a workflow ensemble on shared resources. Overall, $C_{dedicated}$ exhibits the lowest value of the objective function in most cases, which indicates an example of poor placement. We also notice a decline of $C_{colocated}$ at high numbers of ensemble members which closes the gap from the other two configurations. This may be due to the congestion of a high number of data requests to the staging server (recall from Section 2.2, the in-memory staging area is implemented by DIMES) as there are numerous concurrent ensemble members communicating data to each other at the same time. We leave the investigation of this behavior for future work. $P_i^{\mathrm{U,A}}$ assists to distinct $C_{colocated}$ from $C_{dedicated}$ and C_{hybrid} as it favors configurations with higher level of data locality. Finally, $P_i^{\mathrm{U,A,P}}$ groups executions by the number of compute nodes utilized, so that the performance evaluation considers the resource cost defining by node count. The remark is consistent as $C_{colocated}$ still offers the highest objective value among configurations for a given number of ensemble members.

6 | RELATED WORK

Modern scientific workflows commonly feature multiple coupled components, which need to be monitored at the same time to understand the global performance of the workflow. Recent monitoring systems for scientific workflows use system-level information to extract insights into the execution of the workflows. LDMS[25] developed distributed profiling services to periodically sample resource utilization metrics of compute nodes running the workflow. SOS[26] leverages conventional HPC monitoring tools to build an online performance profile that can be run alongside the workflow execution to analyze workflow behaviors. However, traditional performance tools are not designed for modern workflows featuring in situ processing. They collect potentially unnecessary data and may incur significant overhead during profiling.

Several works have addressed monitoring overhead by introducing their particular methods to evaluate a subset of desired features of the workflows. Taufer et al.[27] leveraged domain-specific metrics such as lost frames to characterize in situ analytic tasks using various job mappings. Zacarias et al.[28] estimated the performance degradation arising from co-located applications using a machine learning model. SeeSAw[29] maximized the performance of in situ analysis under power constraints using energy management approaches. WOWMON[30] implemented a runtime that provides a monitoring scheme for scientific workflows composed of in situ tasks by collecting a set of proposed metrics, and a machine learning-based performance diagnosis to validate if the collected metrics are necessary or redundant. While these works focused on in situ workflows, evaluating the performance of the workflow ensembles is not a straightforward extension of evaluating individual workflows. Our work defines the performance of ensembles of in situ workflows.

Ensemble-based methods[3,4] recently gained attention in the computational science, mainly due to the growth of computing power of large-scale systems allowing more simulations to run in parallel. Ensembles are an efficient approach for enhancing sampling techniques, exploring broader configuration space and overcoming the local minima problem observed in scientific simulations. Multiple-walker[3] allowed faster convergence and better sampling by exploiting multiple replicas that simultaneously explore free-energy landscape in addition to transition coordinates of the system. Generalized ensembles[4] explored multiple states of a simulation in ensembles with a probability weight factor so that a random walk in a particular state can escape the energy barrier.

Several recent efforts attempted to efficiently manage the execution of ensemble-based simulations combined with analysis tasks. John et al.[31] proposed a workflow management system that stores task provenances to enable adaptive ensemble simulation. EnTK[12] is a general-purpose toolkit that abstracts components and tasks in an ensemble-based workflow to support various scenarios in which the number of tasks or task dependencies can vary. These works build on RADICAL-Pilot as runtime system[32]. However, these works focus on workflow ensembles with traditional data coupling among tasks and not on workflow ensembles of in situ tasks like the proposed work. A recent study[33] has aimed to prepare the HPC software stack to sustain concurrent execution of multiple simulations and in situ analyses.

7 | CONCLUSION

In this paper, we characterize an ensemble of in situ workflows using multiple configurations and placements. Based on the insights gained from this characterization, we introduce a theoretical framework that models the execution of workflow ensembles when multiple simulations are coupled with multiple analyses using in situ techniques. We define the notion of computational efficiency for workflow ensembles at component level, and then extend this notion to member and ensemble levels by designing several performance indicators. These indicators capture the performance of a workflow ensemble by aggregating several metrics of the given workflow ensemble in terms of resource usage efficiency and resources allocated for components, members, and the entire ensemble. By evaluating these indicators on a real molecular dynamic simulation use case, we show the advantages of data locality when co-locating the simulation with the corresponding analyses in an ensemble member. This finding allows us to schedule each ensemble member of the workflow ensemble individually on a distinct allocation, targeting the co-location among ensemble components of each ensemble member.

Future work will consider leveraging the proposed indicators for scheduling in situ components of a workflow ensemble under resource constraints. The performance indicators appear to be beneficial to assisting in the comparison between different scheduling decisions to optimize scientific discovery. Another future work direction is adapting our performance framework to more complex domain-specific use cases of workflow ensembles, e.g. adaptive sampling[12] in which simulations are periodically executed and restarted.

ACKNOWLEDGEMENTS

This work is funded by NSF contracts #1741040, #1741057, and #1841758; and DOE contract #DE-SC0012636. This research used resources of NERSC, a U.S. DOE Office of Science Facility operated under contract #DE-AC02-05CH11231 and OLCF at the Oak Ridge National Laboratory supported by the Office of Science in the U.S. DOE under contract #DE-AC05-00OR22725.

References

1. Malawski M, Juve G, Deelman E, Nabrzyski J. Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. *Future Generation Computer Systems* 2015; 48: 1-18.

2. Cheatham III TE, Roe DR. The Impact of Heterogeneous Computing on Workflows for Biomolecular Simulation and Analysis. *Computing in Science Engineering* 2015; 17(2).

3. Comer J, Phillips JC, Schulten K, Chipot C. Multiple-Replica Strategies for Free-Energy Calculations in NAMD: Multiple-Walker Adaptive Biasing Force and Walker Selection Rules. *Journal of Chemical Theory and Computation* 2014; 10(12).

4. Chelli R, Signorini GF. Serial Generalized Ensemble Simulations of Biomolecules with Self-Consistent Determination of Weights. *Journal of Chemical Theory and Computation* 2012; 8(3).

5. Vetter JS, Brightwell R, Gokhale M, et al. Extreme Heterogeneity 2018 - Productive Computational Science in the Era of Extreme Heterogeneity: Report for DOE ASCR Workshop on Extreme Heterogeneity. 2018.

6. Khoshlessan M, Paraskevakos I, Fox GC, Jha S, Beckstein O. Parallel performance of molecular dynamics trajectory analysis. *Concurrency and Computation: Practice and Experience* 2020; 32.

7. Ferreira da Silva R, Filgueira R, Pietri I, Jiang M, Sakellariou R, Deelman E. A Characterization of Workflow Management Systems for Extreme-Scale Applications. *Future Generation Computer Systems* 2017; 75: 228–238. doi: 10.1016/j.future.2017.02.026

8. Zhang F, Jin T, Sun Q, et al. In-memory staging and data-centric task placement for coupled scientific simulation workflows. *Concurrency and Computation: Practice and Experience* 2017; 29(12).

9. Ferreira da Silva R, Callaghan S, Do TMA, Papadimitriou G, Deelman E. Measuring the impact of burst buffers on data-intensive scientific workflows. *Future Generation Computer Systems* 2019; 101. doi: 10.1016/j.future.2019.06.016

10. Mondragon OH, Bridges PG, Levy S, Ferreira KB, Widener P. Understanding Performance Interference in Next-Generation HPC Systems. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*; 2016.

11. Do TMA, Pottier L, Ferreira da Silva R, Caíno-Lores S, Taufer M, Deelman E. Assessing Resource Provisioning and Allocation of Ensembles of In Situ Workflows. In: *50th International Conference on Parallel Processing Workshop*; 2021; New York, NY, USA

12. Balasubramanian V, Jensen T, Turilli M, Kasson P, Shirts M, Jha S. Adaptive Ensemble Biomolecular Applications at Scale. *SN Computer Science* 2020; 1(2): 104.

13. Dauwe D, Friese R, Pasricha S, Maciejewski A, Koenig G, Siegel H. Modeling the Effects on Power and Performance from Memory Interference of Co-located Applications in Multicore Systems. In: *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*; 2014.

14. Akyuz N, Georgieva ER, Zhou Z, et al. Transport domain unlocking sets the uptake rate of an aspartate transporter. *Nature* 2015; 518(7537): 68–73.

15. Bjelkmar P, Larsson P, Cuendet MA, Hess B, Lindahl E. 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.* 2010; 6(2).

16. Johnston T, Zhang B, Liwo A, Crivelli S, Taufer M. In situ data analytics and indexing of protein trajectories. *Journal of Computational Chemistry* 2017; 38(16).

17. Taufer M, Estrada T, Johnston T. A survey of algorithms for transforming molecular dynamics data into metadata for *in situ* analytics based on machine learning methods. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 2020; 378(2166): 20190063.

18. Barducci A, Bonomi M, Parrinello M. Metadynamics. *WIREs Computational Molecular Science* 2011; 1(5).

19. NERSC, Lawrence Berkeley National Laboratory's Supercomputer Cori. https://www.nersc.gov/users/computational-systems/cori; .

20. Shende SS, Malony AD. The Tau Parallel Performance System. *The International Journal of High Performance Computing Applications* 2006; 20(2).

21. Do TMA, Pottier L, Caíno-Lores S, et al. A lightweight method for evaluating in situ workflow efficiency. *Journal of Computational Science* 2021; 48. doi: 10.1016/j.jocs.2020.101259

22. Do TMA, Pottier L, Thomas S, et al. A Novel Metric to Evaluate In Situ Workflows. In: *Computational Science – ICCS 2020*; 2020; Cham: 538–553

23. Oxley MA, Jonardi E, Pasricha S, et al. Rate-based thermal, power, and co-location aware resource management for heterogeneous data centers. *Journal of Parallel and Distributed Computing* 2018; 112: 126-139.

24. Zou P, Feng X, Ge R. Contention Aware Workload and Resource Co-Scheduling on Power-Bounded Systems. In: *2019 IEEE International Conference on Networking, Architecture and Storage (NAS)*; 2019: 1-8.

25. Agelastos A, Allan B, Brandt J, et al. The Lightweight Distributed Metric Service: A Scalable Infrastructure for Continuous Monitoring of Large Scale Computing Systems and Applications. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*; 2014: 154–165.

26. Wood C, Sane S, Ellsworth D, et al. A Scalable Observation System for Introspection and in Situ Analytics. In: *Proceedings of the 5th Workshop on Extreme-Scale Programming Tools*; 2016; Salt Lake City, Utah: 42–49.

27. Taufer M, Thomas S, Wyatt M, et al. Characterizing In Situ and In Transit Analytics of Molecular Dynamics Simulations for Next-Generation Supercomputers. In: *15th International Conference on eScience (eScience)*; 2019.

28. Zacarias FV, Petrucci V, Nishtala R, Carpenter P, Mossé D. Intelligent Colocation of Workloads for Enhanced Server Efficiency. In: *2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*; 2019; Campo Grande, Brazil: 120-127.

29. Marincic I, Vishwanath V, Hoffmann H. SeeSAw: Optimizing Performance of In-Situ Analytics Applications under Power Constraints. In: *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*; 2020; New Orleans, LA, USA: 789-798.

30. Zhang X, Abbasi H, Huck K, Malony AD. WOWMON: A Machine Learning-based Profiler for Self-adaptive Instrumentation of Scientific Workflows. *Procedia Computer Science* 2016; 80: 1507-1518.

31. Ossyra J, Sedova A, Tharrington A, Noé F, Clementi C, Smith JC. Porting Adaptive Ensemble Molecular Dynamics Workflows to the Summit Supercomputer. In: Weiland M, Juckeland G, Alam S, Jagode H. , eds. *High Performance Computing*Springer International Publishing; 2019; Cham: 397–417.

32. Merzky A, Turilli M, Maldonado M, Santcroos M, Jha S. Using Pilot Systems to Execute Many Task Workloads on Supercomputers. In: *Job Scheduling Strategies for Parallel Processing*; 2019.

33. Peterson JL, Bay B, Koning J, et al. Enabling machine learning-ready HPC ensembles with merlin. *Future Generation Computer Systems* 2022.