Assessing Resource Provisioning and Allocation of Ensembles of In Situ Workflows

Tu Mai Anh Do
Information Sciences Institute
University of Southern California
Marina Del Rey, CA, USA
tudo@isi.edu

Loïc Pottier
Information Sciences Institute
University of Southern California
Marina Del Rey, CA, USA
lpottier@isi.edu

Rafael Ferreira da Silva
Information Sciences Institute
University of Southern California
Marina Del Rey, CA, USA
rafsilva@isi.edu

Silvina Caíno-Lores
University of Tennessee at Knoxville
Knoxville, TN, USA
scainolo@utk.edu

Michela Taufer
University of Tennessee at Knoxville
Knoxville, TN, USA
mtaufer@utk.edu

Ewa Deelman
Information Sciences Institute
University of Southern California
Marina Del Rey, CA, USA
deelman@isi.edu

ABSTRACT

Scientific breakthroughs in biomolecular methods and improvements in hardware technology have shifted from a single long-running simulation to a large set of shorter simulations running simultaneously, called an ensemble. In an ensemble, each independent simulation is usually coupled with several analyses that apply identical or distinct algorithms on data produced by the corresponding simulation. Today, in situ methods are used to analyze large volumes of data generated by scientific simulations at runtime. This work studies the execution of ensemble-based simulations paired with in situ analyses using in-memory staging methods. Because simulations and analyses forming an ensemble typically run concurrently, deploying an ensemble requires efficient co-location-aware strategies, making sure the data flow between simulations and analyses that form an in situ workflow is efficient. 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 the complex behaviors of ensembles. Thus, we propose a method to evaluate the performance of ensembles of workflows that captures resource usage (efficiency), resource allocation, and component placement. Experimental results demonstrate that our proposed method can effectively capture the performance of different component placements in an ensemble. By evaluating different co-location scenarios, our performance indicator demonstrates improvements of up to four orders of magnitude when co-locating simulation and coupled analyses within a single computational host.

KEYWORDS

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

ACM Reference Format:
Tu Mai Anh Do, Loïc Pottier, Rafael Ferreira da Silva, Silvina Caíno-Lores, Michela Taufer, and Ewa Deelman. 2021. Assessing Resource Provisioning and Allocation of Ensembles of In Situ Workflows. In *50th International Conference on Parallel Processing Workshop (ICPP Workshops '21), August 9–12, 2021, Lemont, IL, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3458744.3474051

1 INTRODUCTION

Organizing computations into ensembles is gaining popularity in many scientific domains using computational simulations. 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 [18]. Workflow ensembles are often used in molecular dynamics (MD) simulations, which compute the atomic states of a molecular system evolving over time by observing microscopic interactions between atoms. Studying the folding process of complex molecules (i.e., conformational transition) of a molecular system often requires running large-scale simulations to thoroughly explore feasible solutions in the configuration space. Such simulations require considerable computing time and resources, which may grow exponentially with the size of the system. Such simulations are often run on high-performance computing (HPC) systems in parallel [9]. Ensemble-based simulation approaches (in which multiple simulations are run concurrently) may also potentially lead to more efficient sampling of the solution space. For instance, multiple-walker [11, 24] employs multiple replicas of the system, known as walkers, where each walker simultaneously explores the same free energy landscape to improve sampling performance. Generalized ensembles [10, 22] allow sampling a broader configuration space by partitioning simulation states into ensembles with optimal weights to perform a random walk in potential energy spaces. The key challenge for enabling these approaches on large-scale systems is to efficiently execute these concurrent simulations structured as an entity, an ensemble.

Traditionally, MD simulations and the follow on data analysis are loosely coupled, where the analysis is started after the simulation is completed. The coupling of 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 [27], post-processing of potentially large volume of simulation data results in I/O bottlenecks [17]. In situ processing has emerged as an alternative paradigm to overcome such I/O limitation. 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 [30], or fast local storage such as burst buffers [14] and doing the analysis in an iterative manner. 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, the simulations are coupled with analyses by staging data in memory for in situ processing.

To denote a collection of workflows, two terms co-exist in the literature: *workflow ensemble* [8, 15] and *ensemble workflow* [3, 23]. Although these terms are used interchangeably, we only refer to *workflow ensemble* in this work. When running ensembles of in situ workflows, there is a tension between co-locating simulations, corresponding analyses on the same resources, so that the data flowing between them can be efficiently communicated, and leveraging separate resources for these components to reduce the computation time of each (as running multiple components on the same resource usually leads to performance degradation due to interference [21]). In this paper, we have developed methods to characterize the execution of the workflow ensemble and to decide how the workflow components need to be place within a system to optimize the overall workflow ensemble performance. We introduce a set of performance metrics that qualify and quantify the contention between components sharing the same computing allocation and the benefits of the co-location.

Commonly, an ensemble-based simulation is comprised of a large number of components. Solely observing individual components separately is not sufficient to characterize the execution of a workflow ensemble, which features concurrently running executables that utilize in situ communication techniques. The heterogeneous behaviors of coupled tasks, i.e. simulations are normally compute-intensive while analyses are data-intensive, exacerbate the management to accommodate efficient execution and make performance characterization of workflow ensembles challenging. Managing the execution of workflow ensembles leads to scheduling challenges at multiple levels within the workflow ensemble, among both concurrent and coupled applications. In this work, we aim to design a method that will allow scientists to make efficient scheduling decisions for a workflow ensemble of coupled simulations and in situ analyses. In particular, 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. Our contributions are as follows:

(1) We introduce a set of comprehensive metrics that can characterize the overall workflow ensembles behavior at different levels of the application (task, workflow, and ensemble). Experimental analysis using a real-world MD in situ workflows demonstrates the usefulness of the approach.
(2) We propose a formal execution model to capture workflow ensemble execution, which is then used to compute the efficiency of coupled components. This formal framework lays out the foundation for a novel performance indicator, which allows us to assess the expected efficiency of a given configuration of a workflow ensemble.
(3) We validate our proposed metrics using a realistic MD use case executing on a leadership class system. Experimental results demonstrate that our methods can capture co-location scenarios in which improvements up to four orders of magnitude can be achieved.

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 platforms. 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 developed new metrics that can better capture 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 [10, 11, 22, 24]. In this work, we restrict ourselves to a single simulation per ensemble member. This simulation is coupled with at least one analysis component. In this work, 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 [5]). 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 read.

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 much better 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

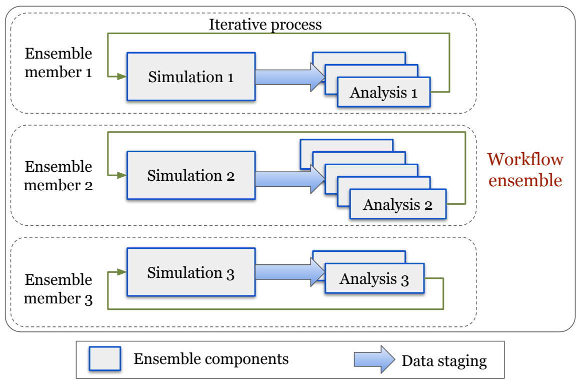

Figure 1: Ensemble of in situ workflows: Ensemble Component, Ensemble Member, and Workflow Ensemble.

architecture). However, this co-location can also lead to resource contention and 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 [12] 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 taking the difference between the end time of the latest analysis and the start time of the simulation. The ensemble makespan is defined as the maximum makespan of 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 [4]. Molecular interactions are implemented in GROMACS [7], 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] as a collective variable [6] 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 [30], burst-buffers [14], 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 using the DTL plugin to write out data abstracted into a *chunk*, which is the base data representation manipulated 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 does 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.

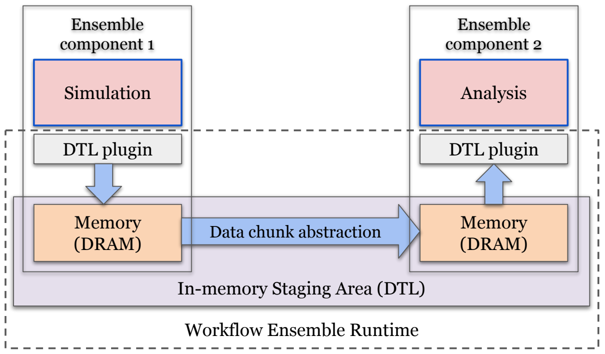

Figure 2: Architectural overview of the proposed runtime system for managing workflow ensemble executions. The Data Transport Layer (DTL) represents in-memory staging area, and the DTL plugins provide the interface between the ensemble components and the underlying DTL.

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 [1], 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 and 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 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 800 and 30, 000 simulation steps and each analysis uses 8 physical cores.

We leverage DIMES [30] 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 [25] to collect runtimes, performance counters, and memory footprints. Measurements are averaged over 5 trials.

Workflow configurations. In this work, we experiment with an workflow ensembles 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 2 ensemble members explore a number of 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.

Configuration	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

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. $C1.1$

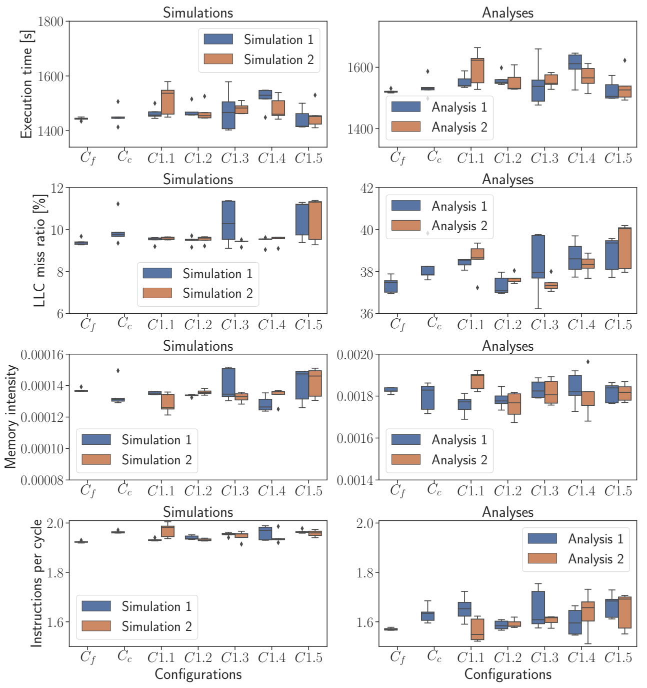

Figure 3: Metrics at ensemble component level.

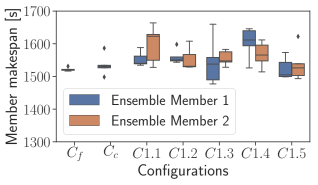

Figure 4: Ensemble member makespan.

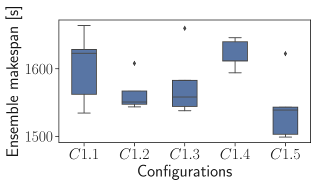

Figure 5: Workflow ensemble makespan.

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.

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 [13] to include multiple analysis components coupled to a single simulation (Figure 1). We 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: a simulation stage S, an idle stage I^S, and 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: a reading stage R, an analyzing stage A, and 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: computational stages (S, A), I/O stages (W, R), and 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 this work, in conformity with [13]). 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.

3.2 In Situ Step

A given ensemble member is composed of a single simulation Sim coupled with K analyses $Ana^1, Ana^2, \dots, Ana^K$. 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 $\{(Sim, Ana^1), \dots, (Sim, Ana^K)\}$ 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 durations, 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 runtime (Idle Simulation scenario) then, the non-overlapped step is equals to $R^i_* + A^i_*$. Hence,

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

Given the non-overlapped segment of in situ steps, we compute the execution time of one ensemble member (also known as the makespan) as:

$$\textsc{Makespan} = n_{steps} \times \overline{\sigma}_* , \tag{2}$$

where n_{steps} is the total number of in situ steps.

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, and as a result increase 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 to be the average time of effective computation over the actual in situ step of K couplings in the ensemble member:

$$E = \frac{1}{K} \sum_{i=1}^{K} \left(1 - \frac{I^S_* + I^{A_i}_*}{\overline{\sigma}_*} \right) = \frac{S_* + W_*}{\overline{\sigma}_*} + \frac{\sum_{i=1}^{K} A^i_* + R^i_*}{K\,\overline{\sigma}_*} - 1. \tag{3}$$

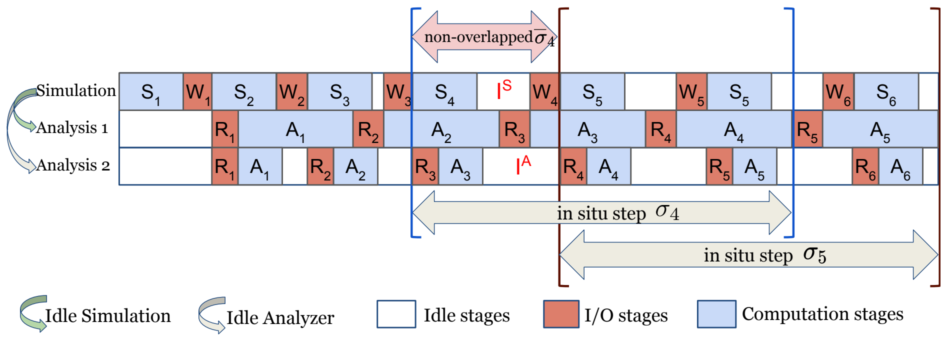

Figure 6: Example of fine-grained execution steps for a member of one ensemble. (Idle simulation and analyzer represent coupled simulation-analysis scenarios.)

Since this indicator is derived from $\overline{\sigma}_*$, which is used to estimate the makespan, maximizing E implies minimizing the idle time and thereby the makespan.

3.4 Discussion

In this section, we use our efficiency model to substantiate the choice of settings (i.e, number of cores and stride) used to run the experiments shown in Section 2.2. Recall that for that set of experiments, we consider a MD simulation coupled with an 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 finds 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 – thus narrowing the configuration space.

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 other co-location scenarios. Based on our first assumption, we arbitrarily set the settings of a simulation as follows: 16 cores and a stride of 800 (recall that our execution platform has compute nodes embedding 32 cores). We then vary the number of cores allocated to the analyses to determine for which number of cores the makespan is minimized and the computational efficiency E is maximized.

We notice that minimizing the makespan is equivalent to minimizing $\overline{\sigma}_*$ (Equation (2)). Thus, given an ensemble member with a certain simulation with a predefined configuration coupled with in situ analyses, in order to minimize the makespan, we need to assign a number of cores the to the analyses such that:

$$R_*^i + A_*^i \leq S_* + W_*, \forall i \in \{1, 2, \ldots, K\}. \quad (4)$$

This inequality implies that each of the K coupling (Sim, Ana^i) falls into the Idle Analyzer scenario, then from Equation (1) we obtain $\overline{\sigma}_* = S_* + W_*$. 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 to 4 cores takes longer than the simulation step, i.e. $R_* + A_* > S_* + W_*$, thus $\overline{\sigma}_* = R_* + A_*$. The inequality in Equation (4) is satisfied once the analysis uses between 8 and 32 cores, which minimizes $\overline{\sigma}_*$, thereby minimizing the member makespan. Among executions whose makespan is minimized, we optimize the computation efficiency by selecting the configuration that leads to $max(E)$. Hence, we decide to assign 8 cores to each analysis, which results in the highest computational efficiency, i.e. the smallest amount of idle time.

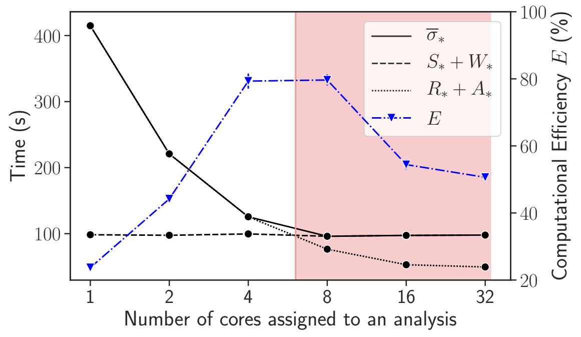

Figure 7: Execution time of the in situ step and computational efficiency when varying the number of cores assigned to an analysis with a fixed simulation setting.

4 PERFORMANCE INDICATORS

In this section, we leverage the efficiency indicator described above to evaluate the performance of workflow ensembles. We extend the notion of efficiency from the individual member to the workflow ensemble level. (Note that different ensemble members can have different efficiency values.) To synthesize the performance of workflow ensembles, we introduce a three-stage approach in which each stage adds a new layer of information to further refine the indicator with desired features such as resource usage, resource provisioning, and resource allocation. The goal of this multi-stage approach is to provide a methodology to assess the impact of each layer of information and come up with an overall performance indicator that can characterize the performance of the entire workflow ensemble. Below, we define a set of notations (Table 3) used to define the indicator.

4.1 Notations

Given a workflow ensemble with N ensemble members $\{EM_1, \dots, EM_N\}$, let P_i be the performance indicator of the ensemble member EM_i , and E_i be its computational efficiency. The ensemble member EM_i contains a simulation Sim_i coupled with K_i analyses, $Ana_i^1, \dots, Ana_i^{K_i}$, thus EM_i has K_i couplings (Sim_i, Ana_i^j), where $j \in \{1, \dots, K_i\}$. 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 nodes used by the workflow ensemble
	Ensemble Member
EM_i	Ensemble member 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 3: Notations.

4.2 Member Resource Usage (U)

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). The first stage P_i^{U} of the performance indicator calculation models the efficiency of an ensemble member in terms of resource usage. We define P_i^{U} as the smallest unit of efficiency in terms of single core usage. Precisely, P_i^{U} computes the ratio between the the computational efficiency E_i of an ensemble member EM_i and the total number of cores c_i used by EM_i, then:

$$P_i^{\text{U}} = \frac{E_i}{c_i}. \tag{5}$$

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 is spent on computing (in contrast to idling), thus the ensemble member makespan is reduced.

4.3 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 $P_i^{\text{U,A}}$ to quantify the benefit 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:

$$CP_i = \frac{1}{K_i}\left(\frac{|s_i|}{|s_i \cup a_i^1|} + \dots + \frac{|s_i|}{|s_i \cup a_i^{K_i}|}\right) = \frac{|s_i|}{K_i} \sum_{j=1}^{K_i} \frac{1}{|s_i \cup a_i^j|}. \tag{6}$$

Intuitively, CP_i describes the placement of EM_i. It decreases with the number of computing nodes used for a given workflow ensemble. $CP_i = 1$ indicates that the EM_i components are all co-located, and CP_i close to 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 in the next stage of the performance indicator. Specifically, we multiply the first stage of our performance indicator by the corresponding placement indicator as follows:

$$P_i^{\text{U,A}} = P_i^{\text{U}} \times CP_i = \frac{E_i}{c_i} \frac{|s_i|}{K_i} \sum_{j=1}^{K_i} \frac{1}{|s_i \cup a_i^j|}. \tag{7}$$

Deriving from the discussed insight of 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.

4.4 Ensemble Resource Provisioning (P)

Finally, by just considering the execution features at the level of ensemble member 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 a different number of computing nodes, the run using a smaller number of nodes should yield better efficiency in two settings with the same performance. Therefore, to obtain the last stage $P_i^{\text{U,A,P}}$, we weigh the performance indicator by the total number of compute nodes M so that the number of resources provisioned for the entire workflow ensemble is considered:

$$P_i^{\mathrm{U,A,P}} = \frac{P_i^{\mathrm{U,A}}}{M} = \frac{E_i}{c_i M} \frac{|s_i|}{K_i} \sum_{j=1}^{K_i} \frac{1}{|s_i \cup a_i^j|}. \quad (8)$$

5 EXPERIMENTAL EVALUATION

In this section, we evaluate the ability of the proposed performance indicators to characterize the execution performance of workflow ensembles. First, we propose a method for aggregating indicator values from individual ensemble members into a global indicator at the workflow ensemble level. Then, we extend our previous experimental configuration settings (Section 2.2) with scenarios in which multiple analyses are coupled with the simulation.

5.1 Ensemble-level Performance Indicator

In order to compute a global indicator, we synthesize performance indicators of every ensemble member. A simple approach would be to consider the average values for all P_i. However, the large variation between these values may lead to 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:

$$F(P_i) = \overline{P} - \sqrt{\frac{1}{N}\sum_{i=1}^{N}(P_i - \overline{P})^2} \quad \text{where} \quad \overline{P} = \frac{1}{N}\sum_{i=1}^{N} P_i. \quad (9)$$

The intuition behind Equation (9) is to favor workflow ensemble's configurations with good makespan, i.e. configurations with a 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 $F(P_i)$. The higher the value of the objective function, the better the performance of the entire workflow regarding efficiency, makespan, resource usage, and component placement.

5.2 Results and Analysis

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 components. 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 configurations in this set, each ensemble member is a single coupling of a simulation and an in situ analysis. A second set is comprised of configurations ranging from $C2.1$ to $C2.8$ (Table 4). For configurations in this set, the simulation of each ensemble member is coupled with two analyses. 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 $F(P_i)$ for 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 3) 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 4.3. 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}, $P_i^{\mathrm{U,P}}$, $P_i^{\mathrm{U,A}}$, $P_i^{\mathrm{U,P,A}}$, and $P_i^{\mathrm{U,A,P}}$, and assess the ability of our indicator to accurately assess the performance of different co-location configurations.

Results. Figure 8 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 8 left), a new layer is added, either P in the middle top figure or A on the middle bottom to form the next stage. On the contrary of $P_i^{\mathrm{U,A}}$, $P_i^{\mathrm{U,P}}$ 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 good member makespans due to the contention of co-location between two analyses. With $P_i^{\mathrm{U,A,P}}$, we observe the characterization where the performance of $C1.4$ is degraded to lower than $C1.5$, but higher than $C1.1$, $C1.2$, $C1.3$. 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 makespans. $C1.5$ outperforms other configurations, which also validate the common intuition behind 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 [30], in which data resides on the memory of the simulation node, co-locating the analysis having data coupling with such the simulation can be beneficial from data locality to shorten the time of staging data.

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. More number of analyses involved in an ensemble member complicates the performance evaluation using traditional metrics. The fact of utilizing the whole cores of compute nodes in several configurations, e.g. $C2.6$, $C2.7$, $C2.8$, likely saturates the resources, which brings difficulties in comparing them with other configurations where compute nodes are not entirely occupied by ensemble components. This situation motivates the need for a performance indicator able to elect the best potential configuration in terms of efficiency of the workflow ensemble. Figure 9 shows the values taken by the objective function when instantiated with different configurations in the second set. In this case, $P_i^{\mathrm{U,P}}$ 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, $P_i^{\mathrm{U,P,A}}$ 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 $C2.6$, $C2.7$

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 4: Experimental configurations with two ensemble members, each ensemble member has two analyses per simulation.

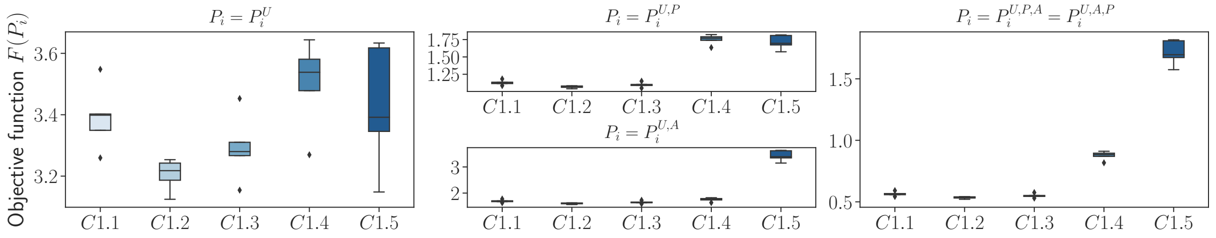

Figure 8: $F(P_i)$ on different P_i orders over configurations which have one analysis per simulation (the higher the better).

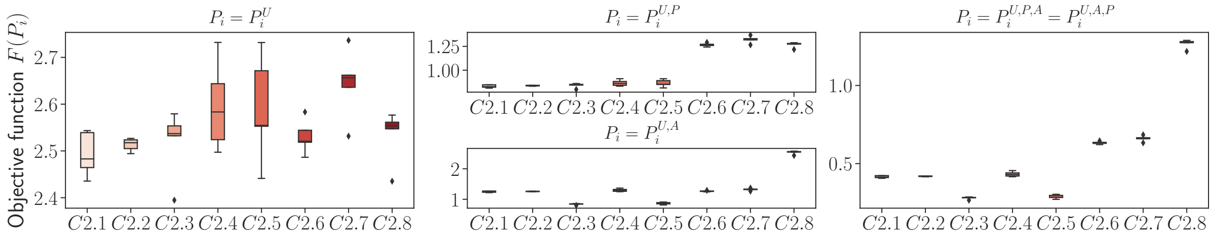

Figure 9: $F(P_i)$ on different P_i orders over configurations which have two analyses per simulation (the higher the better).

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.

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 [2] developed distributed profiling services to periodically sample resource utilization metrics of compute nodes the workflow runs on. SOS [28] relied on conventional HPC monitoring tools [25] to build an online characterization 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 incur potentially significant overheads of 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. [26] leveraged domain-specific metrics such as lost frames to characterize in situ analytic tasks using various job mappings. Zacarias et al. [29] estimated the performance degradation arising from co-located applications using a machine learning model. SeeSAw [19] maximized the performance of in situ analysis under power constraints using energy management approaches. WOWMON [31] 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 [10, 11, 22, 24] 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 [11, 24] allowed faster convergence and better sampling by exploiting multiple replicas that simultaneously explore free-energy landscape along with transition coordinates of the system. Generalized ensembles [10, 22] 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. [23] proposed a workflow management system that stores task provenances to enable adaptive ensemble simulation. EnTK [5] 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. Both of these works rely on RADICAL-Pilot as a runtime system [20]. However, these works study workflow ensembles with traditional data coupling among tasks (i.e., non in situ) while, in this paper, we focus on ensembles workflows comprising in situ tasks.

7 CONCLUSION

In this paper, we have characterized an ensemble of in situ workflows using multiple configurations and placements. Based on the insights gained from this characterization, we have introduced a theoretical framework that models the execution of workflow ensembles when multiple simulations are coupled with multiple analyses using in situ techniques. We have then defined the notion of efficiency for workflow ensembles at component, member, and ensemble levels, and we designed several performance indicators. These indicators capture the performance of 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 have shown 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, worrying only about 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.

ACKNOWLEDGMENTS

This work is funded by NSF contracts #1741040 and #1741057; and DOE contract #DE-SC0012636. This research used resources of NERSC, a U.S. Department of Energy Office of Science User Facility operated under Contract No. DE-AC02-05CH11231.

REFERENCES

[1] 2021. NERSC, Lawrence Berkeley National Laboratory's Supercomputer Cori. https://www.nersc.gov/users/computational-systems/cori
[2] Anthony Agelastos et al. 2014. 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 (SC '14)*. 154–165.
[3] Dong H Ahn et al. 2020. Flux: Overcoming scheduling challenges for exascale workflows. *Future Generation Computer Systems* 110 (2020), 202–213.
[4] Nurunisa Akyuz et al. 2015. Transport domain unlocking sets the uptake rate of an aspartate transporter. *Nature* 518, 7537 (2015).
[5] Vivek Balasubramanian et al. 2020. Adaptive Ensemble Biomolecular Applications at Scale. *SN Computer Science* 1, 2 (2020), 104.
[6] Alessandro Barducci et al. 2011. Metadynamics. *WIREs Computational Molecular Science* 1, 5 (2011).
[7] P Bjelkmar et al. 2010. Implementation of the CHARMM Force Field in GROMACS: Analysis of Protein Stability Effects from Correction Maps, Virtual Interaction Sites, and Water Models. *J. Chem. Theory Comput.* 6, 2 (2010).
[8] S. Caíno-Lores et al. 2020. Applying big data paradigms to a large scale scientific workflow: Lessons learned and future directions. *Future Generation Computer Systems* 110 (2020), 440–452.
[9] T. E. Cheatham III et al. 2015. The Impact of Heterogeneous Computing on Workflows for Biomolecular Simulation and Analysis. *Computing in Science Engineering* 17, 2 (2015).
[10] Riccardo Chelli et al. 2012. Serial Generalized Ensemble Simulations of Biomolecules with Self-Consistent Determination of Weights. *Journal of Chemical Theory and Computation* 8, 3 (2012).
[11] Jeffrey Comer et al. 2014. Multiple-Replica Strategies for Free-Energy Calculations in NAMD: Multiple-Walker Adaptive Biasing Force and Walker Selection Rules. *Journal of Chemical Theory and Computation* 10, 12 (2014).
[12] Daniel Dauwe et al. 2014. 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*. WorldComp.
[13] Tu Mai Anh Do et al. 2021. A lightweight method for evaluating in situ workflow efficiency. *Journal of Computational Science* 48 (2021).
[14] Rafael Ferreira da Silva et al. 2019. Measuring the impact of burst buffers on data-intensive scientific workflows. *Future Generation Computer Systems* 101 (2019).
[15] Q. Jiang et al. 2015. Executing Large Scale Scientific Workflow Ensembles in Public Clouds. In *2015 44th International Conference on Parallel Processing*. IEEE, Beijing, China, 520–529.
[16] Travis Johnston et al. 2017. In situ data analytics and indexing of protein trajectories. *Journal of Computational Chemistry* 38, 16 (2017).
[17] Mahzad Khoshlessan et al. 2020. Parallel performance of molecular dynamics trajectory analysis. *Concurrency and Computation: Practice and Experience* 32 (2020).
[18] Maciej Malawski et al. 2015. Algorithms for cost- and deadline-constrained provisioning for scientific workflow ensembles in IaaS clouds. *Future Generation Computer Systems* 48 (2015), 1–18. Special Section: Business and Industry Specific Cloud.
[19] I. Marincic et al. 2020. SeeSAw: Optimizing Performance of In-Situ Analytics Applications under Power Constraints. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, New Orleans, LA, USA, 789–798.
[20] Andre Merzky et al. 2019. Using Pilot Systems to Execute Many Task Workloads on Supercomputers. In *Job Scheduling Strategies for Parallel Processing*, Dalibor Klusáček, Walfredo Cirne, and Narayan Desai (Eds.).
[21] Oscar H. Mondragon et al. 2016. Understanding Performance Interference in Next-Generation HPC Systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '16)*. Article 33, 12 pages.
[22] Yuko Okamoto. 2004. Generalized-ensemble algorithms: enhanced sampling techniques for Monte Carlo and molecular dynamics simulations. *Journal of Molecular Graphics and Modelling* 22, 5 (2004).
[23] John Ossyra et al. 2019. Porting Adaptive Ensemble Molecular Dynamics Workflows to the Summit Supercomputer. In *High Performance Computing*.
[24] Paolo Raiteri et al. 2006. Efficient Reconstruction of Complex Free Energy Landscapes by Multiple Walkers Metadynamics. *The Journal of Physical Chemistry B* 110, 8 (2006), 3533–3539. PMID: 16494409.
[25] Sameer S. Shende et al. 2006. The Tau Parallel Performance System. *The International Journal of High Performance Computing Applications* 20, 2 (2006).
[26] M. Taufer et al. 2019. Characterizing In Situ and In Transit Analytics of Molecular Dynamics Simulations for Next-Generation Supercomputers. In *15th International Conference on eScience (eScience)*.
[27] Jeffrey S. Vetter et al. 2018. *Extreme Heterogeneity 2018 - Productive Computational Science in the Era of Extreme Heterogeneity: Report for DOE ASCR Workshop on Extreme Heterogeneity*. Technical Report. Lawrence Berkeley National Lab.(LBNL).
[28] Chad Wood et al. 2016. A Scalable Observation System for Introspection and in Situ Analytics. In *Proceedings of the 5th Workshop on Extreme-Scale Programming Tools (ESPT '16)*. IEEE Press, Salt Lake City, Utah, 42–49.
[29] F. V. Zacarias et al. 2019. Intelligent Colocation of Workloads for Enhanced Server Efficiency. In *2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*. IEEE, Campo Grande, Brazil, 120–127.
[30] Fan Zhang et al. 2017. In-memory staging and data-centric task placement for coupled scientific simulation workflows. *Concurrency and Computation: Practice and Experience* 29, 12 (2017).
[31] Xuechen Zhang et al. 2016. WOWMON: A Machine Learning-based Profiler for Self-adaptive Instrumentation of Scientific Workflows. *Procedia Computer Science* 80 (2016), 1507–1518. International Conference on Computational Science 2016, ICCS.