SN Computer Science (2020) 1:104
https://doi.org/10.1007/s42979-020-0081-1

REVIEW ARTICLE

Adaptive Ensemble Biomolecular Applications at Scale

Vivek Balasubramanian[1] · Travis Jensen[2] · Matteo Turilli[1] · Peter Kasson[3] · Michael Shirts[2] · Shantenu Jha[4]

Received: 21 August 2019 / Accepted: 24 February 2020 / Published online: 28 March 2020

Abstract
Recent advances in both theory and methods have created opportunities to simulate biomolecular processes more efficiently using adaptive ensemble simulations. Ensemble-based simulations are used widely to compute a number of individual simulation trajectories and analyze statistics across them. Adaptive ensemble simulations offer a further level of sophistication and simulation efficacy by enabling high-level algorithms to control simulations based on intermediate results. Novel high-level algorithms for adaptive simulations require sophisticated approaches to manage the ensemble members and utilize the intermediate data during runtime. Thus, there is a need for scalable software systems to support adaptive ensemble-based methods. We describe the operations in executing adaptive workflows, classify different types of adaptations, and describe challenges in implementing them in software tools. We establish the design considerations of software systems to support the requirements of adaptive ensemble applications at extreme scale. We use Ensemble Toolkit (EnTK) and its associated task execution runtime system (RADICAL-Pilot)—middleware building blocks to implement a scalable adaptive ensemble execution system. We implement two high-level adaptive ensemble algorithms—multiwalker expanded ensemble and Markov state modeling, and execute up to 2^{12} ensemble members, on thousands of cores on three distinct HPC platforms. We highlight scientific advantages enabled by the novel capabilities of our approach. To the best of our knowledge, this is the first attempt at describing and implementing multiple adaptive ensemble workflows using a common conceptual and implementation framework.

Keywords Adaptive ensembles · Advanced sampling algorithms · High performance computing · Workflow building blocks

Introduction

Current computational methods for solving scientific problems in biomolecular science are at or near their scaling limits using traditional parallel architectures [1]. Computations using straightforward molecular dynamics (MD) are inherently sequential processes, and parallelization is limited to speeding up each individual, serialized, time step. Consequently, *ensemble-based* computational methods have been developed to address these gaps, including replica-exchange molecular dynamics (REMD) [2–8], multiple walker metadynamics [8–10], hyperdynamics and other accelerated dynamics methods [11–13], Markov state modeling [14, 15], and swarm-of-trajectory methods [16–19]. In these methods, multiple simulation tasks are executed concurrently, and various physical or statistical principles are used to combine the tasks together with longer time scale communication (seconds to hours) instead of the microsecond to milliseconds required for standard tightly coupled parallel processing.

Existing ensemble-based methods have been successful for addressing a number of questions in biomolecular modeling [20]. However, studying systems with multiple-timescale behavior extending out to microseconds or milliseconds, or studying even shorter timescales on larger

This article is part of the topical collection "Software Challenges to Exascale Computing" guest edited by Amit Majumdar and Ritu Arora.

✉ Shantenu Jha
shantenu.jha@rutgers.edu

1 Department of ECE, Rutgers University, Piscataway, USA

2 Department of ChBE, University of Colorado Boulder, Boulder, USA

3 Biomedical Engineering, University of Virginia, Charlottesville, USA

4 Department of ECE, Rutgers University and Computational Science Initiative, Brookhaven National Laboratory, Upton, New York, USA

physical systems will not only require tools that can support 100 × −1000× greater degrees of parallelism but also exploration of *adaptive* algorithms. In adaptive algorithms, the intermediate results of simulations are used to alter following simulations. We define *adaptivity* as the capability to change attributes that influence execution performance or domain-specific parameters, based on runtime information.

Adaptive approaches can increase physical simulation efficiency by greater than a thousand-fold [12, 15, 21–24]. Adaptive algorithms learn from the simulation ensemble members as they proceed, "steering" execution toward interesting phase space or parameters and thus improve sampling quality or sampling rate. Adaptivity can access events that would otherwise happen at much longer time scales, making it possible to investigate larger physical systems with a given set of resources as well as to efficiently explore high-dimensional energy surfaces in finer detail. As molecular simulations are used to investigate questions of increasing biological complexity with progressively increasing scales, gains in algorithmic sophistication and computational efficiency from adaptive ensemble methods will become critical in generating quantitative insight into biological problems.

However, such adaptive ensemble simulations require a sophisticated software infrastructure to encode, modularize, and execute complex interactions and execution logic [25–27]. The execution trajectory of adaptive simulations cannot be fully determined *a priori*, but depends upon intermediate results. The logic to specify such changes can rely on a simulation within an ensemble, an operation across an ensemble, or external criteria, such as resource availability or experimental data. To achieve scalability and efficiency, such adaptivity cannot be performed via user intervention and hence automation of the control logic and execution becomes critical.

Many adaptive algorithms can be expressed at a high level, such that the adaptive logic itself is independent of simulation details (i.e., external to MD engines like AMBER [28], NAMD [29] or GROMACS [30]). Adaptive operations that are expressed independent of the internal details of simulation tasks facilitate MD software package agnosticism and simpler expression of different types of adaptivity and responses to adaptivity. Further, it is important to formulate adaptive capabilities so as to be agnostic of the type and execution properties of the analysis responsible for adaptivity. For example, machine learning-based analysis will provide increasingly sophisticated adaptive ensemble algorithms. The separation of adaptive operations from simulation and analysis internals provides a useful abstraction for both methods' developers and software systems. This promotes easy development of new methods while facilitating scalable system software and its optimization through performance engineering [31].

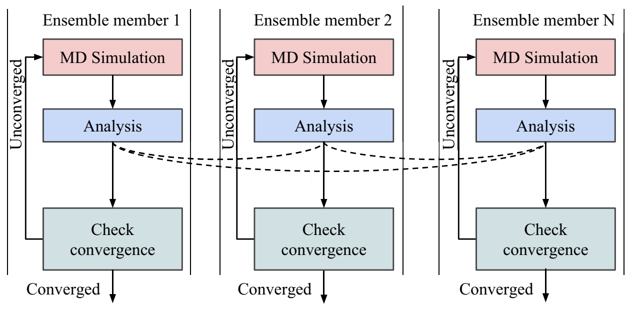

Fig. 1 Schematic of the expanded ensemble (EE) science driver. Two versions of EE are implemented: (1) local analysis where analysis uses only data local to its ensemble member; and (2) global analysis where analysis uses data from other ensemble members (represented by dashed lines)

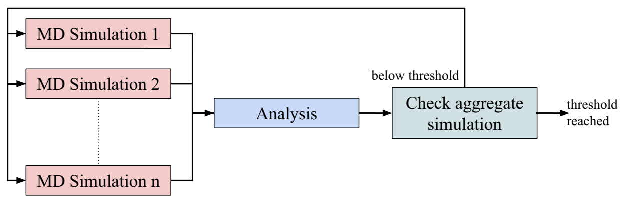

Fig. 2 Schematic of the Markov State Model science driver

In this paper, we focus on the design and implementation of software systems, in particular the Ensemble Toolkit (EnTK) [32] and its associated runtime system (RADICAL-Pilot), to support the requirements of adaptive ensemble applications at extreme scale. To guide the design and implementation of capabilities to encode and execute adaptive ensemble applications in a scalable and adaptive manner, we identify two such applications from the biomolecular science domain, abstract and generalized descriptions of which are shown in Figs. 1 and 2. They have distinct execution requirements; in addition, coordination and communication patterns among their ensemble members differ. However, they are united by their need for an adaptive execution of large number of tasks.

This paper makes the following specific contributions: (i) We describe the operations in executing adaptive workflows, classify different types of adaptations, and describe challenges in implementing them in software tools; (ii) we establish the design considerations of software systems to support the requirements of adaptive ensemble applications at extreme scale. We use middleware building blocks [33, 34] to implement a scalable adaptive ensemble execution system and characterize its performance with respect to adaptive operations; and (iii) we implement two high-level adaptive ensemble algorithms, executing up to 2^{12} ensemble members, on thousands of cores on three distinct high-performance computing (HPC) platforms.

Section "Related Work" describes existing and related approaches. Section "Science Drivers" presents two science drivers that motivate the need for large-scale adaptive ensemble biomolecular simulations. We discuss different types and challenges in supporting adaptivity in "Software Design Considerations for Adaptive Ensemble Workflows" section. In "Ensemble Toolkit" section, we describe the design and implementation of EnTK, and the enhancements made to address the challenges of adaptivity. In "Experiments" section, we characterize the overheads in EnTK as a function of adaptivity types, validate the implementation of the science drivers, and discuss scientific advantages that the novel capabilities of our approach provides.

Related Work

Adaptive ensemble applications span several science domains including, but not limited to, climate science, seismology, astrophysics, and biomolecular science. For example, Ref. [35] studies adaptive selection and tuning of dynamic recurrent neural networks (RNNs) for hydrological forecasting; Ref. [36] presents adaptive modeling of oceanic and atmospheric circulation; Ref. [37] studies adaptive assessment methods on an ensemble of bridges subjected to earthquake motion; and Ref. [38] discusses parallel adaptive mesh refinement techniques for astrophysical and cosmological applications.

Several adaptive ensemble algorithms have been formulated. In generalized ensemble simulation methods, different ensemble simulations employ distinct exchange algorithms [39] or specify diverse sampling parameters [40] to explore free energy surfaces that are less accessible to non-adaptive methods. Weighted ensemble and forward-flux sampling approaches adaptively trim and clone ensemble members using criteria based on progress along a desired collective variable [26, 27]. Markov State Model [41] (MSM) approaches adaptively select initial configurations for simulations to reduce uncertainty of the resulting model.

Current solutions to encode and execute adaptive ensemble algorithms fall into two categories: workflow systems that do not fully support adaptive algorithms, or MD software packages where the adaptivity is embedded within the executing kernels. Several workflow systems [42], including Kepler, Taverna and Pegasus support adaptation capabilities only as a form of fault tolerance and not as a way to enable decision-logic for changing the workflow at runtime. Domain-specific workflow systems such as Copernicus [43] have also been developed to support Markov state modeling algorithms to study kinetics of bio-molecules. Although Copernicus provides an interactive and customized interface to domain scientists, it requires users to manage the acquisition of resources, the deployment of the system, and the configuration of the execution environment. This hinders Copernicus uptake, often requiring tailored guidance from its developers.

Widely used MD software packages such as AMBER [28], NAMD [29] and GROMACS [30] offer capabilities to execute ensemble algorithms, often with some adaptive capability. Encoding the adaptive ensemble algorithm, including its adaptation logic within MD software packages locks the capabilities in those packages, prevents easy addition of new adaptive algorithms or reuse across packages. In contrast, the capability to encode the algorithm and adaptation logic as a high-level workflow promises several benefits: separation between algorithm specification and execution; flexible and quick prototyping of alternative algorithms; and extensibility of algorithmic solutions to multiple software packages, science problems and scientific domains [31, 44]. To realize these promises, we develop the abstractions and capabilities to encode adaptivity at the ensemble application level, while reusing existing capabilities to execute adaptive ensemble applications at scale on high-performance computing (HPC) systems.

Science Drivers

In this paper, we discuss two representative adaptive ensemble applications from the biophysical domain: expanded ensemble and Markov state modeling. Prior to discussing the implementation of these applications, we describe the underlying algorithms.

Expanded Ensemble

Metadynamics [45] and expanded ensemble (EE) dynamics [46] are a class of adaptive simulation algorithms, used in both biological and other condensed matter simulations, where similar to replica exchange, individual simulations jump between simulation conditions. In EE dynamics, the simulation states take one of N discrete 'states' or ensembles of interest, while preserving the probability distribution corresponding to each of the states that would be obtained if that ensemble was simulated alone. These N states can be different temperatures or biasing functions on the system or force field parameters of the system. Metadynamics is similar, except the different simulation states are described by one or more continuous variables. In both algorithms, unlike replica exchange, each simulation can explore the N different simulation states independently. Since some states are inherently more physically probable than others, simulation weights assigned to each state (for EE) or continuously assigned as a function of the simulation variable (metadynamics) are required to force the simulations to visit desired distributions in the simulation condition space, which necessarily requires sampling in all the simulation states while the allowed simulation configurations are also sampled. These weights are learned adaptively as the simulation progresses using a variety of techniques [46].

Since the movement among state spaces is essentially diffusive, the larger the simulation state spaces, the more time the sampling between states takes. "Multiple walker" approaches can improve sampling performance by using more than one simulation to explore the same state space [47]. Further, the simulation condition range can be partitioned into individual simulations as smaller partitions decrease diffusive behavior [9]. The "best" partitions to spend time sampling may not be known until after simulation. These partitions could instead be determined adaptively, based on runtime information about partial simulation results.

To our knowledge, EE simulations have not been performed using a multiwalker approach, in large part because of the difficulty in implementing such a workflow, as the theory itself is very is similar to multiple walker metadynamics. In this paper, we use this framework to implement two versions of EE consisting of concurrent and iterative ensemble members that analyze data at regular intervals. In the first version, we analyze data local to each ensemble member; in the second version we analyze data global to all the ensemble members by asynchronously exchanging data among members. In our application, each ensemble member consists of two types of task: simulation and analysis. The simulation tasks generate MD trajectories while the analysis tasks use these trajectories to generate simulation condition weights for the next iteration of simulation in its own ensemble member. Every analysis task operates on the current snapshot of the total local or global data. Note that in global analysis, EE uses any and all data available and does not explicitly "wait" for data from other ensemble members at the same iteration. Figure 1 is a representation of these implementations.

Markov State Modeling

Markov state modeling (MSM) is another important class of molecular simulation algorithms for determining kinetics of molecular models. Using an assumption of separation of time scales of molecular motion, the rates of first-order kinetic processes are learned adaptively. In a MSM simulation, a large ensemble of simulations, typically tens or hundreds of thousands, are run from different starting points and similar configurations are clustered as states. MSM building techniques include kinetic information but begin with a traditional clustering method (e.g., k-means or k-centers) using a structural metric. Configurations of no more than 2Å to 3Å RMSDs are typically clustered into the same "microstate" [48].

The high degree of structural similarity implies a kinetic similarity, allowing for subsequent kinetic clustering of micro-states into larger "macro-states". The rates of transitions among these states are estimated by observing which entire kinetic behavior can be inferred, even though individual simulations perform no more than one state transition. However, the choice of where new simulations are initiated to best refine the definition of the states, improve the statistics of the rate constants, and discover new simulation states requires a range of analyses of previous simulation results, making the entire algorithm highly adaptive.

MSM provides a way to encode dynamic processes such as protein folding into a set of metastable states and transitions among them. In computing MSM from simulation trajectories, the metastable state definitions and the transition probabilities have to be inferred. Refs. [49, 50] show that "adaptive sampling" can lead to more efficient MSM construction as follows: provisional models are constructed using intermediate simulation results, and these models are then used to direct the placement of further simulation trajectories. Different from other approaches, in this paper we encode this algorithm as an application where the adaptive code is independent from the software packages used to perform the MD simulations and MSM construction.

Figure 2 offers a diagrammatic representation of the adaptive ensemble MSM approach. The application consists of an iterative pipeline with two stages: (i) ensemble of simulations and (ii) MSM construction to determine optimal placement of future simulations. The first stage generates sufficient amount of MD trajectory data for an analysis. The analysis–i.e., the second stage–operates over the cumulative trajectory data to adaptively generate a new set of simulation configurations, used in the next iteration of the simulations. The pipeline is iterated until the resulting MSM converges.

Software Design Considerations for Adaptive Ensemble Workflows

The broad range of adaptive ensemble simulation algorithms impose diverse requirements on the underlying software infrastructure. Algorithms differ in the frequency of communication between ensemble members, local versus non-local communication, and the type of information exchanged. Adaptive changes can alter the number of tasks being performed (how many ensemble members in a simulation), the parameters of those tasks (placement of temperature or lambda values in an expanded-ensemble simulation), or even which tasks are being performed when. The logic to specify such changes can rely on a single simulation within an ensemble, an operation across an ensemble, or even external criteria, such as new experimental data.

Execution of Adaptive Workflows

Adaptive ensemble applications discussed in "Science Drivers" section involve two computational layers: at the lower level each simulation or analysis is performed via MD software package; at the higher level, an **algorithm** codifies the coordination and communication among simulations and between simulations and analyses. Different adaptive ensemble applications and adaptive algorithms might have varying coordination and communication patterns, yet are amenable to common adaptations and similar types of adaptations.

We implement each simulation and analysis instance of these applications as a **task**, while representing the full set of task dependencies as task graph of a **workflow**. A workflow may be fully specified a priori, or may be adapted, changing in specification, during runtime. For the remainder of the paper, we refer to alterations in the task graph as workflow adaptivity.

Executing adaptive workflows at scale on HPC resources presents several challenges [31]. Execution of adaptive workflows can be decomposed into four operations as represented in Fig. 3: (a) creation of an initial task graph, encoding known tasks and dependencies; (b) traversal of the initial task graph to identify tasks ready for execution in accordance with their dependencies; (c) execution of those tasks on the compute resource; and (d) notification of completed tasks (control-flow) or generation of intermediate data (dataflow) which invokes adaptations of the task graph.

Operations (b)–(d) are repeated until the complete workflow is determined, and all its tasks are executed. This sequence of operations is called an Adaptivity Loop: in an adaptive scenario, the workflow "learns" its future task graph based on the execution of its current task graph; in a pre-defined scenario, the workflow's task graph is fully specified and only operations (a)–(c) are necessary.

Encoding of adaptive workflows requires two sets of abstractions: one to encode the workflow; and the other to encode the adaptation methods (A) that, upon receiving a signal x, operate on the workflow. The former abstractions are required for creating the task graph, i.e., operation (a), while the latter are required to adapt the task graph, i.e., operation (d).

Types of Adaptations

Adaptivity Loop applies an adaptation method (Fig. 3d) to a task graph. We represent a task graph as $G = [V, E]$, with the set V of vertices denoting the tasks of the workflow and their properties (such as executable, required resources, and required data), and the set E of directed edges denoting the dependencies among tasks. For a workflow represented as task graph $G^T = [V, E]$, there exist four

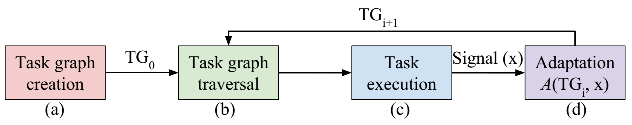

Fig. 3 **Adaptivity Loop**: Sequence of operations in executing an adaptive workflow

parameters that may change during execution: (i) set of vertices; (ii) set of edges; (iii) size of the vertex set; and (iv) size of the edge set. We analyzed the 2^4 permutations of these four parameters and identified 3 that are valid and unique. The remaining permutations represent conditions that are either not possible to achieve, or combinations of the 3 valid permutations.

Task-count adaptation: We define an operator A_C to represent the adaptation of task-count if, on receiving a signal x, the method performs the following adaptation (operation) on G^T:

$$G^T_{i+1} = A_C(G^T_i, x)$$
$$\implies size(V_i) \neq size(V_{i+1}) \wedge size(E_i) \neq size(E_{i+1})$$
$$\text{where } G^T_i = [V_i, E_i] \wedge G^T_{i+1} = [V_{i+1}, E_{i+1}].$$

Task-count adaptation changes the number of tasks, i.e., the adaptation method operates on G^T_i to produce a different G^T_{i+1}, such that at least one vertex and one edge is added or removed to/from G^T_i.

Task-order adaptation: We define an operator A_O as a task-order adaptation if, on a signal x, it performs the following adaptation on G^T:

$$G^T_{i+1} = A_O(G^T_i, x)$$
$$\implies E_i \neq E_{i+1} \wedge V_i = V_{i+1}$$
$$\text{where } G^T_i = [V_i, E_i] \wedge G^T_{i+1} = [V_{i+1}, E_{i+1}].$$

Task-order adaptation changes the dependency order among tasks, i.e., A_O operates on G^T_i to produce G^T_{i+1} such that the vertices are unchanged but at least one of the edges between vertices is different between G^T_i and G^T_{i+1}.

Task-property adaptation: We define an operator A_P that captures the adaptation of the property of tasks, if, on a signal x, it performs the following adaptation on G^T:

$$G^T_{i+1} = A_P(G^T_i, x)$$
$$\implies V_i \neq V_{i+1} \wedge size(V_i) = size(V_{i+1}) \wedge E_i = E_{i+1}$$
$$\text{where } G^T_i = [V_i, E_i] \wedge G^T_{i+1} = [V_{i+1}, E_{i+1}].$$

Task-property adaptation changes the properties of at least one task, i.e., A_P operates on a G^T_i to produce a new G^T_{i+1} such that the edges and the number of vertices are unchanged, but the properties of at least one vertex are different between G_i^T and G_{i+1}^T.

We can represent the workflow of the two science drivers using the notations presented. Expanded ensemble (EE) consists of N ensemble members executing independently for multiple iterations until convergence (meaning the bias weights on each ensemble member are no longer changing to within some tolerance) is reached in any ensemble member. We represent one iteration of each ensemble members as a task graph G^T and the convergence criteria with x. An adaptive EE workflow can then be represented as:

```
parallel_for i in [1 : N] :
    while (condition on x) :
        $G_{i+1}^T = A_P(A_O(A_C(G_i^T)))$
```

Markov state modeling (MSM) consists of one ensemble member which iterates between simulation and analysis till sufficient trajectory data are analyzed. At each analysis step, a set of promising molecular configurations is selected as initial configurations for the next iteration of ensembles. The choice of which configurations are considered "promising" will vary between application and MSM variant [15, 51]. We represent one iteration of the ensemble member as a task graph G^T and its termination criteria as x. An adaptive MSM workflow can then be represented as:

```
while (condition on x) :
    $G^{T+1} = A_O(A_C(G^T))$
```

Challenges in Encoding Adaptive Workflows

Supporting adaptive workflows poses three main challenges. The first challenge is the expressibility of adaptive workflows as their encoding requires APIs that enable the description of the initial state of the workflow and the specification of how the workflow adapts on the basis of intermediate signals. The second challenge is determining when and how to instantiate the adaptation. Adaptation is described at the end of the execution of tasks wherein a new task graph is generated. Different strategies can be employed for the instantiation of the adaptation [52]. The third challenge is the implementation of the adaptation of the task graph at runtime. We divide this challenge into three parts: (i) propagation of adapted task graph to all components; (ii) consistency of the state of the task graph among different components; and (iii) efficiency of adaptive operations.

Ensemble Toolkit

Expressing adaptive algorithms as computational processes separate from but operating on independent ensemble members, creates several implementation challenges. These include coordination and consistency across distributed execution components, scalable communication between independent simulations and efficient stop and restart of simulations.

Separating the adaptive logic from underlying execution management software allows the complexity to be contained within the internal implementation of the software system and not be exposed to the user. This approach also enables transparent low-level optimization and adjustment to fluctuations in workload and resource availability. Thus, scalable ensemble-based adaptive algorithms require support at multiple levels: programming models and APIs, execution models and runtime system.

In this section, we discuss the design and implementation of EnTK and its associated runtime system, as well as enhancements to EnTK to support adaptivity as a first-class capability. We offer a schematic representation of the components and sub-components of EnTK (Fig. 4), summarizing its design and implementation. Further, we detail the enhancements made to EnTK to support the encoding and execution of the three types of adaptation discussed in "Types of Adaptations" section.

Design

EnTK is an ensemble execution system, implemented as a Python library, that offers components to encode and execute ensemble workflows on HPC systems. EnTK decouples the description of ensemble workflows from their execution by separating three concerns: (i) specification of tasks and resource requirements; (ii) resource selection and acquisition; and (iii) management of task execution. EnTK sits between the user and the HPC system, abstracting resource and execution management complexities from the user.

The design, implementation and performance of EnTK are discussed in detail in Refs. [32, 53]. EnTK exposes an API with three user-facing constructs to describe an

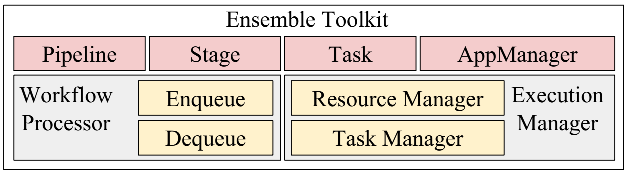

Fig. 4 Schematic of EnTK representing its components and sub-components

ensemble: Pipeline, Stage, and Task. We define the constructs as:

- **Task:** an abstraction of a computational process consisting of the specification of an executable, software environment, resource and data requirement.
- **Stage:** a set of tasks without mutual dependencies that, therefore, can be concurrently executed.
- **Pipeline:** a sequence of stages such that any stage i can be executed only after stage $i-1$.

Ensemble workflows are described by the user as a set or sequence of pipelines, where each pipeline is a list of stages, and each stage is a set of tasks. A set of pipelines executes concurrently, whereas a sequence executes sequentially. All the stages of each pipeline execute sequentially, and all the tasks of each stage execute concurrently. In this way, we describe a workflow in terms of the concurrency and sequentiality of tasks, without requiring the explicit specification of task dependencies.

AppManager is the core component of EnTK, serving two broad purposes: (i) exposing an API to accept the encoded workflow and a specification of the resource requirements from the user; and (ii) managing the execution of the workflow on the specified resource via several components and a third-party runtime system (RTS). AppManager abstracts complexities of resource acquisition, task and data management, heterogeneity, and failure handling from the user. All components and sub-components of EnTK communicate via a dedicated messaging system that is set up by the AppManager.

AppManager instantiates a WorkflowProcessor, which is responsible for maintaining the concurrent and sequential execution of tasks as described by the pipelines and stages in the workflow. WorkflowProcessor consists of two components, Enqueue and Dequeue, that are used to: enqueue sets of executable tasks, i.e., tasks with all their dependencies satisfied; and dequeue executed tasks, to and from dedicated queues.

AppManager also instantiates an ExecutionManager, which is responsible for managing the resources and the execution of tasks on these resources. ExecutionManager consists of two sub-components: ResourceManager and TaskManager. Both sub-components interface with a RTS to manage the allocation and deallocation of resources, and the execution of tasks, received via dedicated queues, from the WorkflowProcessor.

EnTK manages failures of tasks, components, computing infrastructure (CI), and RTS. Depending on user configuration, failed tasks can be resubmitted or ignored. EnTK, by design, is resilient against components failure as all state updates are transactional: failed components can simply be re-instantiated. Both the CI and RTS are considered black boxes, and their partial failures are assumed to be handled locally. Upon full failure of the CI or RTS, EnTK assumes all the resources and the tasks undergoing execution are lost, restarts the RTS, and resumes execution from the last successful pipeline, stage, and task.

Implementation

EnTK is implemented in Python and uses the RabbitMQ message queuing system [54] and the RADICAL-Pilot (RP) [55] task execution RTS. All EnTK components are implemented as processes, and all sub-components as threads. AppManager is the master process spawning all the other processes. Tasks, stages and pipelines are implemented as objects, copied among processes and threads via queues and transactions. Process synchronization uses message-passing via queues.

Using RabbitMQ offers several benefits: (i) producers and consumers are unaware of topology, because they interact only with the server; (ii) messages are stored in the server and can be recovered upon failure of EnTK components; (iii) messages can be pushed and pulled asynchronously because data can be buffered by the server upon production; and (iv) $\geq O(10^6)$ tasks are supported.

EnTK uses RADICAL-Pilot (RP) as the RTS. RP is a pilot system, i.e., a middleware component that enables the submission of "pilot" jobs to the resource manager of an HPC platform. The defining capability of pilot systems is the decoupling of resource acquisition from task execution. These systems allow for queuing a single job on the HPC platform and, once this job becomes active, they enable the direct scheduling of tasks on the acquired resources, without waiting in the HPC platform's queue. Pilot systems do not 'game' the resource manager of the HPC platform: Once queued, jobs are managed according to the platform's policies. RP provides access to several HPC systems, including XSEDE, ORNL, and NCSA resources, and can be configured to use other HPC systems.

Once integrated, EnTK and RP form an end-to-end system for: (1) describing an ensemble application; (2) acquiring HPC resources; (3) scheduling tasks of the ensemble application on those resources; and (4) executing those tasks respecting their priority relationship. This integrated system uses a multi-level, multi-entity scheduling algorithm. Initially, a job is scheduled on the HPC platform to acquire resources; then, tasks that can be executed concurrently (i.e., a workload) are scheduled by EnTK on RP that, in turn, schedules them as compute units into an Agent that was bootstrapped on the HPC resources. Agent is responsible for scheduling compute units on available resources, placing these units onto specific nodes, cores or GPUs, and launching these units for execution.

Usually, EnTK and RP acquire all the resources needed to execute the whole workflow before starting its execution. Resource acquisition consists in submitting a job to the HPC platform, while resource release involves killing the job once the entire workflow has been executed. Thus, users wait in the HPC platform's batch system only once before executing their workflow. When the resources have been acquired, RP binds the amount and type of available resources needed by each compute unit at execution time, and unbind those resources right after the unit has been executed. Binding and unbinding resources to compute units does not require using the HPC platform's batch system: RP owns the resources for the required walltime and exercises full control over their usage.

EnTK and RP concurrently and sequentially execute compute units on the resources acquired by submitting a job to the HPC platform's batch system. This allows to optimize scheduling algorithms, based on the type and amount of units that need to be executed, and to maximize resource utilization by optimizing both the physical and temporal placement of units on available resources. Note that these capabilities support efficient implementation of adaptivity: depending on runtime conditions, elements of the workflows can be redefined or new elements can be added to the existing workflow. RP controls the amount of concurrency with which the adapted workflow is executed, depending on the amount of resources available.

Note that executing workflows may entail executing separate groups of tasks, each group requiring a different amount of resources. In this case, trade offs must be made between the amount of concurrency of the execution of each group of tasks, their execution time, and the amount of resource utilization throughout the execution of the whole workflow. For example, given a workflow with two groups of tasks A and B, where B must be executed after A, and assuming that the fully concurrent execution of A requires 2048 cores while that of B just 1024, a decision will be made whether to privilege time to execution by requiring 2048 cores or resource utilization by requiring 1024. In the former case, A and B will be executed with maximal concurrency and therefore minimal time to completion; in the latter case, A will be executed with 50% concurrency and, roughly, twice as long execution time. Note that when using maximal concurrency, 1024 cores will idle when executing B, while when using 50% of concurrency for A, all the cores will always be utilized throughout the execution of the workflow.

Enhancements for Adaptive Execution

In "Challenges in Encoding Adaptive Workflows" section, we described three challenges in supporting adaptive workflows: (i) expressibility of adaptive workflows; (ii) when and how to trigger adaptation; and (iii) implementation of adaptive operations. We addressed these challenges by implementing three new capabilities in EnTK: (1) expressing an adaptation operation; (2) executing the operation; and (3) modifying a task graph at runtime.

Adaptations in ensemble workflows follow the Adaptivity Loop described in "Execution of Adaptive Workflows" section. Execution of one or more tasks is followed by some signal x that triggers an adaptation operation. In EnTK, this signal is currently implemented as a control signal triggered at the end of a stage or a pipeline. We added the capability to express this adaptation operation as post-execution properties of stages and pipelines. In this way, when all the tasks of a stage or all the stages of a pipeline have completed, the adaptation operation can be invoked to evaluate whether a change in the task graph is required. This evaluation is based on the results of the ongoing computation and it is performed asynchronously, i.e., without effecting any other executing tasks.

The adaptation operation is encoded as a Python property of the Stage and Pipeline objects. The encoding requires the specification of three functions: one function to evaluate a boolean condition over x, and two functions to describe the adaptation, depending on the result of the boolean evaluation. Users define the three functions specified as post-execution properties of a Stage or Pipeline, based on the requirements of their application. As such, these functions can modify the existing task graph or extend it as per the three adaptivity types described in "Types of Adaptations" section.

Reference [52] specifies multiple strategies to perform adaptation: forward recovery, backward recovery, proceed, and transfer. In EnTK, we implement a non-aggressive adaptation strategy, similar to 'transfer', where a new task graph is created by modifying the current task graph only after the completion of part of that task graph. The choice of this strategy is based on the current science drivers where tasks that have already executed and tasks that are currently executing are not required to be adapted but all forthcoming tasks might be.

Modifying the task graph at runtime requires coordination among EnTK components to ensure consistency in the task graph representation. AppManager holds the global view of the task graph and, upon instantiation, Workflow Processor maintains a local copy of that task graph. The dequeue sub-component of Workflow Processor acquires a lock over the local copy of the task graph, and invokes the adaptation operation as described by the post-execution property of stages and pipelines. If the local copy of the task graph is modified, Workflow Processor transmits those changes to AppManager that modifies the global copy of task graph, and releases the lock upon receiving an acknowledgment. This ensures that adaptations to the task graph are consistent across all components, while requiring minimal communication.

Pipeline, stage, and task descriptions alongside the specification of an adaptation operation as post-execution for pipelines and stages enable the expression of adaptive workflows. The 'transfer' strategy enacts the adaptivity of the task graph, and the implementation in EnTK ensures consistency and minimal communication in executing adaptive workflows. Note how the design and implementation of adaptivity in EnTK does not depend on specific capabilities of the software package executed by each task of the ensemble workflow.

Note that the separation of concern between expressing the adaptive logic of the workflow and executing the scientific code of the workflow's tasks, makes EnTK a general-purpose tool for codifying adaptive ensemble applications. Thus, EnTK can be used to express well-known adaptive algorithms but also to explore new solutions. Reference [56] offers examples of how to implement adaptive code in EnTK while Ref. [57] shows the code implementing the adaptive expanded sample used in our experiments.

Experiments

We perform three sets of experiments. The first set characterizes the overhead of EnTK when performing the three types of adaptation described in "Types of Adaptations" section. The second set validates our implementation of the two science drivers presented in "Science Drivers" section against reference data. The third set compares our implementation of adaptive expanded ensemble algorithm with local and global analysis against results obtained with a single and an ensemble of MD simulations.

We use four application kernels in our experiments: `stress-ng` [58], `GROMACS` [30], `OpenMM` [59] and Python scripts. `stress-ng` allows to control the computational duration of a task for the experiments that characterize the adaptation overhead of EnTK, while `GROMACS` and `OpenMM` are the simulation kernels for the expanded ensemble and Markov state modeling validation experiments.

We executed all experiments from the same host machine, but we targeted three HPC systems, depending on the amount and availability of the resources required by the experiments, and the constraints imposed by the queue policy of each machine. NCSA Blue Waters and ORNL Titan were used for characterizing the adaptation overhead of EnTK, while XSEDE SuperMIC was used for the validation and production scale experiments. When we run our experiments, NCSA Blue Waters had 22500 nodes, each with 32 cores; ORNL Titan had 18688 nodes, each with 16 cores; and XSEDE SuperMIC had 382 nodes, each with 20 cores.

Characterization of Adaptation Overhead

We perform five experiments to characterize the overhead of adapting ensemble workflows encoded using EnTK. Each experiment measures the overhead of a type of adaptation as a function of the number of adaptations. In the case of task-count adaptation, the overhead is measured also as a function of the number of tasks and of their type, single- or multi-node. This is relevant because with increasing size of the simulated molecular system, multi-node tasks may have lower time-to-solution than single-node ones.

Each experiment measures EnTK Adaptation Overhead and Task Execution Time. The former is the time taken by EnTK to adapt the workflow by invoking user-specified algorithms; the latter is the time taken to run the executables of all tasks of the workflow. Consistent with the scope of this paper, the comparison between each adaptation overhead and task execution time offers a measure of the efficiency with which EnTK implements adaptive functionalities. Ref. [53] has a detailed analysis of other overheads of EnTK.

Table 1 describes the variables and fixed parameters of the five experiments about adaptivity overheads in EnTK. In these experiments, the algorithm is encoded in EnTK as 1 pipeline consisting of several stages with a set of tasks. In the experiments I–III about task-count adaptation, the pipeline initially consists of a single stage with 16 tasks of a certain type. Each adaptation, at the completion of a stage, adds 1 stage with a certain number of tasks of a certain type, thereby increasing the task-count in the workflow.

In experiments IV–V, the workflow is encoded as 1 pipeline with 17, 65, or 257 stages with 16 tasks per stage. Each adaptation occurs upon the completion of a stage and, in the case of task-order adaption, the remaining stages of a pipeline are shuffled. In the case of task-property adaption, the number of cores used by the tasks of the next stage is set to a random value below 16, keeping the task type to single-node. The last stage of both experiments is non-adaptive, resulting in 16, 64, and 256 total adaptations.

In the experiments I, IV and V, where the number of adaptations varies, each task of the workflow executes the `stress-ng` kernel for 60 seconds. For the experiments II and III with $O(1000)$ tasks, the execution duration is set to 600 seconds so to avoid performance bottlenecks in the underlying runtime system and therefore interferences with the measurement of EnTK adaptation overheads. All experiments have no data movement as the performance of data operations is independent from that of adaptation.

Figure 5i, iv, v shows that EnTK Adaptation Overhead and Task Execution Time increase linearly with the increasing of the number of adaptations. EnTK Adaptation Overhead increases due to the time taken to compute the additional adaptations and its linearity indicates that the

Table 1 Experiment parameters plotted in Fig. 5

Figure	Adaptation type	Experiment variable	Fixed parameters
I	Task-count	Num. of adaptations	Num. of tasks added per adaptation = 16, Type of tasks added = single-node
II	Task-count	Num. of tasks added per adaptation	Num. of adaptations = 2, Type of tasks added = single-node
III	Task-count	Type of tasks added	Num. of adaptations = 2, Num. of tasks added per adaptation = $2^{10} * 2^{s}$ (s = stage index)
IV	Task-order	Num. of adaptations	Num. of re-ordering op. per adaptation = 1, Type of re-ordering = uniform shuffle
V	Task-property	Num. of adaptations	Property type modified per adaptation = 1, Property adapted = Num. of cores per task

computing time of each adaptation is constant. Task Execution Time increases due to the time taken to execute the tasks of the stages that are added to the workflow as a result of the adaptation.

Figure 5i, iv, v also shows that task-property adaptation (v) is the most expensive, followed by task-order adaptation (iv) and task-count (i) adaptation. These differences depend on the computational cost of the Python functions executed during adaptation: in task-property adaptation, the function parses the entire workflow and invokes the Python `random.randint` function 16 times per adaptation; in task-order adaptation, the Python function shuffles a Python list of stages; and in task-count adaption, the Python function creates an additional stage, appending it to a list.

In Fig. 5ii, EnTK Adaptation Overhead increases linearly with an increase in the number of tasks added per task-count adaptation, explained by the cost of creating additional tasks and adding them to the workflow. The Task Execution Time remains constant at $\approx 1200s$, since sufficient resources are acquired to execute all the tasks concurrently.

Figure 5iii compares EnTK Adaptation Overhead and Task Execution Time when adding single-node and multi-node tasks to the workflow. The former is greater by $\approx 1s$ when adding multi-node tasks, whereas the latter remains constant at $\approx 1200s$ in both scenarios. The difference in the overhead, although negligible when compared to Task Execution Time, is explained by the increased size of a multi-node task description. As in Fig. 5ii, Task Execution Time remains constant due to availability of sufficient resources to execute all tasks concurrently.

Experiments I–V show that EnTK Adaptation Overhead is proportional to the computing required by the adaptation algorithm and is not determined by the design or implementation of EnTK. In absolute terms, EnTK Adaptation Overhead is orders of magnitude smaller than Task Execution Time. Thus, EnTK advances the practical use of adaptive ensemble workflows.

Validation of Science Driver Implementations

We implement the two science drivers of "Science Drivers" section using the abstractions developed in EnTK. We validate our implementation of expanded ensemble (EE) by calculating the binding of the cucurbit[7]uril 6-amino-1-hexanol host-guest system, a molecular recognition system often used for testing as exhaustive simulation can get the right answer on a reasonable timescale [60, 61], and our implementation of Markov state modeling (MSM) by simulating the Alanine dipeptide system and comparing our results with the reference data of the DESRES group [62].

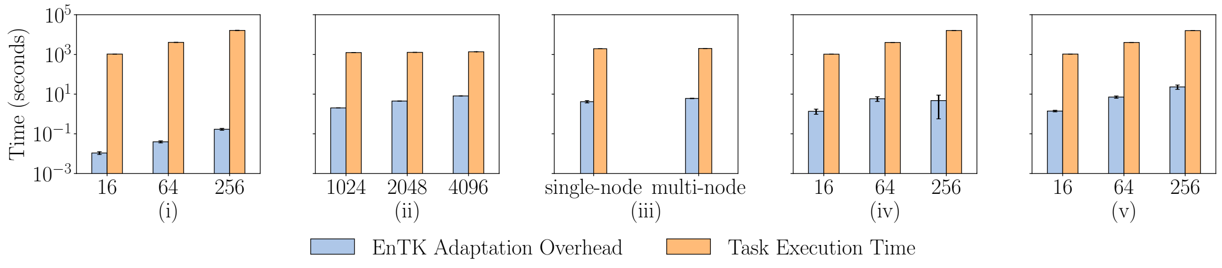

Fig. 5 EnTK Adaptation Overhead and Task Execution Time for task-count (**i**, **ii**, and **iii**), task-order (**iv**), and task-property (**v**) adaptations

Expanded Ensemble

We execute the EE science driver described in "Expanded Ensemble" section on XSEDE SuperMIC for a total of 2270ns MD simulation time using GROMACS 5.1.3 (to match behavior in previous studies). To validate the process, we carry out a set of simulations of the binding of cucurbit[7]uril (host) to 6-amino-1-hexanol (guest) in explicit solvent. Simulation details and sample input files can be found in a previous study Ref [60]. Simulations were run for a total of 29.12 ns per ensemble member. Validation was done by comparing the final free energy estimate to a reference calculation run with a single adaptive expanded ensemble simulation. Each ensemble member is encoded in EnTK as a pipeline of stages of simulation and analysis tasks, where each pipeline uses 1 node for 72 hours. With 16 ensemble members (i.e., pipelines) for the current physical system, we use $\approx 1k$ node hours of computational resources.

The expanded ensemble variable in these simulations is the degree of coupling (i.e., the strength of the energetic interaction term) between the guest and the rest of the system (water and host). As the system explores the coupling parameter using EE dynamics, this strengthening and weakening allows the guest to binds and unbind from the host over the course of the simulation, where if the interactions were left entirely on, the ligand would remain bound during a simulation of this timescale. The free energy of this process is gradually estimated over the course of the simulation, using the Wang–Landau algorithm [63], as implemented in this system as described in Ref. [60]. However, we hypothesize that we can speed convergence by 1) estimate free energies using the potential energy differences among states and the Multistate Bennett Acceptance Ratio (MBAR) algorithm [64] at intermediate steps, treating the expanded ensemble simulations as quasistatic processes, and 2) allowing individual ensemble members to share information with each other about the free energies of the different ensembles rather than using only their own trajectory to estimate it.

We consider four variants of the EE method:

- **Method 1:** one continuous simulation, omitting *any* intermediate analysis using MBAR.
- **Method 2:** multiple parallel simulations without *any* intermediate analysis using MBAR.
- **Method 3:** multiple parallel simulations with local intermediate analysis, i.e., using current and historical simulation information from only its own ensemble member.
- **Method 4:** multiple parallel simulations with global intermediate analysis, i.e., using current and historical simulation information from all ensemble members.

In each method, the latter 2/3 of the simulation data available at the time of each analysis is used for free energy estimates via the MBAR algorithm. In methods 3 and 4, we avoid the use of instantaneous weights due to Wang–Landau algorithm by using all of the quasistatic sampling data to determine the weights using MBAR during the intermediate analyses. These weights provide, in theory, a better estimate of the weights that are used to force simulations to visit desired distributions in the simulation condition space (see "Expanded Ensemble" section). Note that in methods 3 and 4, where intermediate analysis is used to update the weights, the intermediate analysis, external to GROMACS, is always applied at 320ps intervals.

The reference calculation consisted of four parallel expanded ensemble simulations that each ran for 200 ns each with fixed initial weights. These simulations used a set of previously estimated weights, which were themselves from a 400 ns expanded ensemble using the Wang–Landau algorithm (similar to a single member of the ensemble from Method 1, but run for much longer). MBAR was used to estimate the free energy for each of these simulations, which generate the fully stationary probability distribution of the simulation due to fixed, non-adaptive weights. The reference value is reported as the MBAR estimate of the pooled reference data, and its error is reported as the standard deviation of the non-pooled MBAR estimates. This calculation is therefore a reference for correctness, not a control for efficiency.

Figure 6 shows the free energy estimates obtained through each of the four methods with the reference calculation value. Final estimates of each method agree within error to the reference value. Validating that the four methods used to implement adaptive ensembles converge the free energy estimate to the actual value.

Markov State Modeling

We execute the MSM science driver described in "Markov State Modeling" section on XSEDE SuperMIC for a total of 100ns MD simulation time over multiple iterations. Each iteration of the task graph is encoded in EnTK as one pipeline with 2 stages consisting of 10 simulation tasks and 1 analysis task. Each task uses 1 node to simulate 1ns.

We compare the results obtained from execution of the EnTK implementation against reference data by performing the clustering of the reference data and deriving the mean eigenvalues of two levels of the metastable states, i.e., macro- and micro-states. The reference data were generated by a non-adaptive workflow consisting of 10 tasks, each simulating 10ns.

Eigenvalues attained by the macro-states (top) and micro-states (bottom) in the EnTK implementation and reference data are plotted as a function of the state index in Fig. 7. Final eigenvalues attained by the implementation

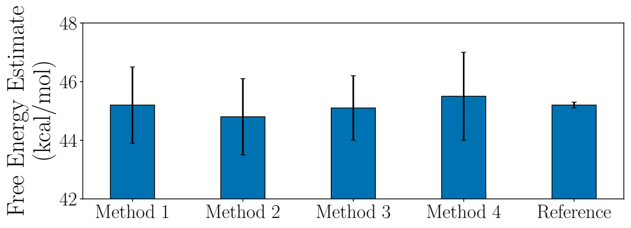

Fig. 6 Validation of EE implementation: Observed variation of free energy estimate for methods 1–4. Reference is the MBAR estimate and standard deviation of four 200ns fixed weight expanded-ensemble simulations

agree with the reference data within the error bounds. The validation of the implementation warrants that similar implementations should be investigated for larger molecular systems and longer durations, where the aggregate duration is unknown and termination conditions are evaluated during runtime.

Evaluation of Methodological Efficiency Using Adaptive Capabilities in EnTK

We analyzed the convergence properties of the free energy estimate using the data generated for the validation of EE. The convergence behavior of Method 1 observed in Fig. 8 suggests that the non-ensemble method converges faster than ensemble-based methods with the same total simulation time. However, it does not necessarily represent the average behavior of the non-ensemble-based approach. The average behavior is depicted more clearly by Method 2 because this method averages the free energy estimate of 16 independent single simulations. The apparent improved convergence may be due to the fact that the simulation is continuous, and can potentially reach configurations not sampled in Method 2, or may simply represent lucky stochastic fluctuations in the weights.

The most significant feature of Fig. 8 is that all three ensemble-based methods converge at similar rates to the reference value. We initially hypothesized that adding adaptive analysis to estimate the weights would improve convergence behavior, but we see no significant change in these experiments. Analysis of these simulations revealed a fundamental physical reason that demonstrates a need for additional adaptivity to successfully accelerate these simulations. Although expanded ensemble simulations allowed the ligand to move in and out of the binding pocket rapidly, the slowest motion, occurring on the order of 10s of nanoseconds, was the movement of water out of the binding pocket, which is needed to allow the ligand to rebind as water backs into a simulation biases that equilibrate on shorter timescales may overly stabilize either configurations of with waters out or waters, preventing the sampling of both configurations. Combining the weights from multiple simulations does not lower the kinetic barriers for the water transition. Additional biasing variables are needed to algorithmically accelerate this slow motions, requiring a combination of metadynamics and expanded ensemble simulations, with biases both in the protein interaction variable and the collective variable of water occupancy in the binding pocket. The same ensemble approach may be more useful with multiple nonphysical dimensions, resulting in a larger space than can be sampled by a single ensemble member.

The methodology described here gives researchers the ability to implement additional adaptive elements and test their effects on system properties. Additionally, as referenced

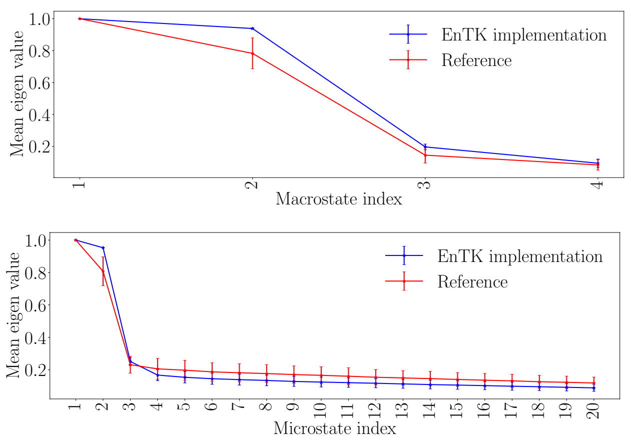

Fig. 7 Mean eigenvalue attained by the macro-states (top) and micro-states (bottom) by Alanine dipeptide after aggregate simulation duration of 100ns implemented using EnTK compared against reference data

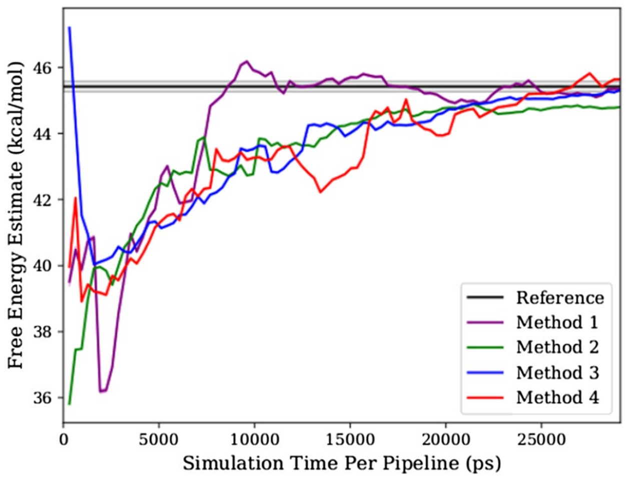

Fig. 8 Convergence of the ensemble expanded ensemble implementation: Observed convergence behavior in Methods 1–4. Reference is the MBAR estimate of the pooled data and the standard deviation of the non-pooled MBAR estimates of four 200ns fixed weight expanded ensemble simulations

in "Enhancements for Adaptive Execution" section, these adaptive elements can be implemented on relatively short time scales, giving the ability to test many implementations, as done in this paper.

Analysis of the slow motions of the system suggests the potential power of more complex and general adaptive patterns. Simulations with accelerated dynamics along the hypothesized degrees of freedom can be carried out, and resulting dynamics can be analyzed, automated and monitored for degrees of freedom associated with remaining slow degrees of motion [65]. Accelerated dynamics can be adaptively adjusted as the simulation process continues. Characterization experiments suggest that EnTK can support the execution of this enhanced adaptive workflow with minimal overhead.

Conclusion

Novel adaptive algorithms and methods across domains such as biomolecular science, climate science and uncertainty quantification leverage intermediate data to study larger problems, longer time scales and to engineer better fidelity in the modeling of complex phenomena. Adaptive ensemble simulations methods provide a promising route to enhancing the computational efficiency of biomolecular simulations over vanilla ensemble MD simulations. As we approach exascale computing, and molecular simulations are used to address questions of increasing biological complexity, gains in algorithmic sophistication and computational efficiency from adaptive ensemble methods will become critical in generating quantitative insight into biological problems.

In this paper, we described the operations in executing adaptive ensemble workflows, classified the different types of adaptations, and described challenges in implementing them in software tools. We discuss how the Ensemble Toolkit was designed to support the scalable the execution of adaptive ensemble workflows. We characterized the adaptation overhead in EnTK, validated the implementation of two science drivers. We executed expanded ensemble at scales and highlight the advantages of using adaptive ensemble capabilities developed here to accelerate methodological advances.

The primary contribution of this work is the design of a software infrastructure that permits new adaptive methods, and ultimately, their applications to important biophysical problems at unprecedented scales. To the best of our knowledge, this is the first attempt at describing and implementing multiple adaptive ensemble workflows using a common conceptual and implementation framework.

Acknowledgements We acknowledge support from NSF 1440677, 1639694 and 1835449. XSEDE computational resources were made available via XRAC allocation TG-MCB090174. On behalf of all authors, the corresponding author states that there is no conflict of interest.

References

1. Cheatham TE, Roe DR. The impact of heterogeneous computing on workflows for biomolecular simulation and analysis. Comput Sci Eng. 2015;17(2):30–9.
2. Trebst S, Troyer M, Hansmann UHE. Optimized parallel tempering simulations of proteins. J Chem Phys. 2006;124:174903.
3. Hansmann UHE. Parallel tempering algorithm for conformational studies of biological molecules. Chem Phys Lett. 1997;281:140–50.
4. Mitsutake A, Sugita Y, Okamoto Y. Replica-exchange multicanonical and multicanonical replica-exchange Monte Carlo simulations of peptides. I. Formulation and benchmark test. J Chem Phys. 2003;118:6664.
5. Mitsutake A, Okamoto Y. Replica-exchange extensions of simulated tempering method. J Chem Phys. 2004;121:2491.
6. Ballard AJ, Jarzynski C. Replica exchange with nonequilibrium switches. Proc Natl Acad Sci. 2009;106(30):12224–9. https://doi.org/10.1073/pnas.0900406106.
7. Rauscher S, Neale C, Pomes R. Simulated tempering distributed replica sampling, virtual replica exchange, and other generalized-ensemble methods for conformational sampling. J Chem Theory Comput. 2009;5(10):2640–62. https://doi.org/10.1021/ct900302n ISSN: 1549-9618.
8. 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. J Chem Theory Comput. 2014;10(12):5276–85 ISSN: 1549-9618.
9. Janosi L, Doxastakis M. Accelerating flat-histogram methods for potential of mean force calculations. J Chem Phys. 2009;131(5):054105 ISSN: 1089-7690.
10. Raiteri P, Laio A, Gervasio FL, Micheletti C, Parrinello M. Efficient reconstruction of complex free energy landscapes by multiple walkers metadynamics. J Phys Chem B. 2006;110:3533–9.
11. Voter AF. Hyperdynamics: accelerating molecular dynamics of infrequent events. Phys Rev Lett. 1997;78:3908–11. https://doi.org/10.1103/PhysRevLett.78.3908.
12. Huang C, Perez D, Voter AF. Hyperdynamics boost factor achievable with an ideal bias potential. J Chem Phys. 2015;143:074113. https://doi.org/10.1063/1.4928636.
13. Voter AF. Parallel replica method for dynamics of infrequent events. Phys Rev B. 1998;57(22):13985–8.
14. Chodera JD, Swope WC, Pitera JW, Dill KA. Long-time protein folding dynamics from short-time molecular dynamics simulations. Multiscale Model Simul. 2006;5(4):1214–26.
15. Bowman GR, Huang X, Pande VS. Using generalized ensemble simulations and Markov state models to identify conformational states. Methods. 2009;. https://doi.org/10.1016/j.ymeth.2009.04.013.
16. Maragliano L, Roux B, Vanden-Eijnden E. Comparison between mean forces and swarms-of-trajectories string methods. J Chem Theory Comput. 2014;10(2):524–33. https://doi.org/10.1021/ct400606c.
17. Atzori A, Bruce NJ, Burusco KK, Wroblowski B, Bonnet P, Bryce RA. Exploring protein kinase conformation using swarm-enhanced sampling molecular dynamics. J Chem Inf Model. 2014;54(10):2764–75. https://doi.org/10.1021/ci5003334.

18. Sanchez-Martinez M, Field M, Crehuet R. Enzymatic minimum free energy path calculations using swarms of trajectories. J Phys Chem B. 2015;119(3):1103–13. https://doi.org/10.1021/jp506593t.
19. Pan AC, Sezer D, Roux B. Finding transition pathways using the string method with swarms of trajectories. J Phys Chem B. 2008;112(11):3432–40.
20. Husic BE, Pande VS. Markov state models: from an art to a science. J Am Chem Soc. 2018;140(7):2386–96.
21. Bowman GR, Ensign DL, Pande VS. Enhanced modeling via network theory: adaptive sampling of markov state models. J Chem Theory Comput. 2010;6(3):787–94.
22. Miron RA, Fichthorn KA. Accelerated molecular dynamics with the bond-boost method. J Chem Phys. 2003;119(12):6210–6. https://doi.org/10.1063/1.1603722.
23. Voter AF. Parallel replica method for dynamics of infrequent events. English. Phys Rev B. 1998;57(22):13985–8.
24. Suárez E, Lettieri S, Zwier MC, Stringer CA, Subramanian SR, Chong LT, Zuckerman DM. Simultaneous computation of dynamical and equilibrium information using a weighted ensemble of trajectories. J Chem Theory Comput. 2014;10(7):2658–67. https://doi.org/10.1021/ct401065r.
25. Dakka J, Balasubramanian KPV, Turilli M, Wright DW, Zasada SJ, Wan S, Coveney PV, Jha S. [n. d.] Concurrent and adaptive extreme scale binding free energy calculations. in review. arXiv:1801.01174.
26. Zwier MC, Adelman JL, Kaus JW, Pratt AJ, Wong KF, Rego NB, Surez E, Lettieri S, Wang DW, Grabe M, Zuckerman DM, Chong LT. Westpa: an interoperable, highly scalable software package for weighted ensemble simulation and analysis. J Chem Theory Comput. 2015;11(2):800–9. https://doi.org/10.1021/ct5010615.
27. DeFever RS, Hanger W, Sarupria S, Kilgannon J, Apon AW, Ngo LB. Building a scalable forward flux sampling framework using big data and hpc. In: Proceedings of the practice and experience in advanced research computing on rise of the machines (Learning) (PEARC'19). ACM, Chicago, IL, USA, 2019;3:1–3:8. ISBN: 978-1-4503-7227-5. https://doi.org/10.1145/3332186.3332205
28. Case DA, Cheatham TE, Darden T, Gohlke H, Luo R, Merz KM, et al. The amber biomolecular simulation programs. J Comput Chem. 2005;26(16):1668–88.
29. Phillips JC, Braun R, Wang W, Gumbart J, Tajkhorshid E, Villa E, et al. Scalable molecular dynamics with namd. J Comput Chem. 2005;26(16):1781–802.
30. Abraham MJ, Murtola T, Schulz R, Páall S, Smith JC, Hess B, Lindahl E. Gromacs: high performance molecular simulations through multi-level parallelism from laptops to supercomputers. SoftwareX. 2015;1:19–25.
31. Kasson PM, Jha S. Adaptive ensemble simulations of biomolecules. Curr Opin Struct Biol. 2018;52:87–94.
32. Balasubramanian V, Treikalis A, Weidner O, Jha S. Ensemble toolkit: scalable and flexible execution of ensembles of tasks. In: 2016 45th international conference on parallel processing (ICPP). Volume 00, 2016;458–463. https://doi.org/10.1109/ICPP.2016.59.
33. Turilli M, Balasubramanian V, Merzky A, Paraskevakos I, Jha S. [n. d.] Middleware building blocks for workflow systems. Computing in Science & Engineering (CiSE) special issue on Incorporating Scientific Workflows in Computing Research Processes. 2019; https://doi.org/10.1109/MCSE.2019.2920048. arXiv:1903.10057.
34. Balasubramanian V, Jha S, Merzky A, Turilli M. Radical-cybertools: middleware building blocks for scalable science. CoRR. 2019; arXiv:1904.03085.
35. Coulibaly P, Baldwin CK. Nonstationary hydrological time series forecasting using nonlinear dynamic methods. J Hydrol. 2005;307(1–4):164–74.
36. Behrens J, Rakowsky N, Hiller W, Handorf D, Läuter M, Päpke J, et al. Amatos: parallel adaptive mesh generator for atmospheric and oceanic simulation. Ocean Model. 2005;10(1–2):171–83.
37. Casarotti C, Pinho R. An adaptive capacity spectrum method for assessment of bridges subjected to earthquake action. Bull Earthq Eng. 2007;5(3):377–90.
38. Lan Z, Taylor VE, Bryan G. Dynamic load balancing for structured adaptive mesh refinement applications. In: International Conference on Parallel Processing, 2001. IEEE, 2001; p. 571–579.
39. Okamoto Y. Generalized-ensemble algorithms: enhanced sampling techniques for monte carlo and molecular dynamics simulations. J Mol Graph Model. 2004;22(5):425–39.
40. Babin V, Roland C, Sagui C. Adaptively biased molecular dynamics for free energy calculations. J Chem Phys. 2008;128(13):134101.
41. Chodera JD, Swope WC, Pitera JW, Dill KA. Long-time protein folding dynamics from short-time molecular dynamics simulations. Multiscale Modeli Simul. 2006;5(4):1214–26.
42. Mattoso M, Dias J, Ocaña KACS, Ogasawara E, Costa F, Horta F, et al. Dynamic steering of hpc scientific workflows: a survey. Future Gen Comput Syst. 2015;46:100–13.
43. Pronk S, Pouya I, Lundborg M, Rotskoff G, Wesen B, Kasson PM, Lindahl E. Molecular simulation work-flows as parallel algorithms: the execution engine of copernicus, a distributed high-performance computing platform. J Chem Theory Comput. 2015;11(6):2600–8.
44. McKinley PK, Sadjadi M, Kasten EP, Cheng BHC. Composing adaptive software. Computer. 2004;37(7):56–64.
45. Barducci A, Bonomi M, Parrinello M. Metadynamics. Wiley Interdiscip Rev Comput Mol Sci. 2011;1(5):826–43. https://doi.org/10.1002/wcms.31.
46. Chelli R, Signorini GF. Serial generalized ensemble simulations of biomolecules with self-consistent determination of weights. J Chem Theory Comput. 2012;8(3):830–42.
47. 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. J Chem Theory Comput. 2014;10(12):5276–85.
48. Pande VS, Beauchamp K, Bowman GR. Everything you wanted to know about markov state models but were afraid to ask. Methods. 2010;52(1):99–105.
49. Singhal N, Pande VS. Error analysis and efficient sampling in markovian state models for molecular dynamics. J Chem Phys. 2005;123(20):204909.
50. Hinrichs NS, Pande VS. Calculation of the distribution of eigenvalues and eigenvectors in markovian state models for molecular dynamics. J Chem Phys. 2007;126(24):244101.
51. Scherer MK, Trendelkamp-Schroer B, Paul F, Perez-Hernandez G, Hoffmann M, Plattner N, Wehmeyer C, Prinz J-H, Noe F. Pyemma 2: a software package for estimation, validation, and analysis of markov models. J Chem Theory Comput. 2015;11(11):5525–42.
52. van der Aalst WMP, Jablonski S. Dealing with workflow change: identification of issues and solutions. Comput Syst Sci Eng. 2000;15(5):267–76.
53. Balasubramanian V, Turilli M, Hu W, Lefebvre M, Lei W, Modrak RT, Cervone G, Tromp J, Jha S. Harnessing the power of many: extensible toolkit for scalable ensemble applications. In: 2018 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2018, Vancouver, BC, Canada, May 2018;21-25, 536–545. https://doi.org/10.1109/IPDPS.2018.00063.
54. [n. d.] Rabbitmq. https://www.rabbitmq.com/ (Accessed 03/2018).
55. Merzky A, Turilli M, Maldonado M, Santcroos M, Jha S. Using pilot systems to execute many task workloads on supercomputers. Job Scheduling Strategies for Parallel Processing - 22nd International Workshop, JSSPP 2018. Vancouver. 2018;2018:61–82. https://doi.org/10.1007/978-3-030-10632-44.

56. Balasubramanian V. https://radicalentk.readthedocs.io/en/latest/advanced_examples.html. (2019).
57. Balasubramanian V. https://github.com/radical-experiments/adap-bms-exps-ipdps18/blob/master/expanded-ensemble/bin/runme.py. 2019.
58. [n. d.] Stress-ng. http://kernel.ubuntu.com/~cking/stress-ng/stress-ng.pdf (accessed March 2018). ().
59. [n. d.] Openmm. https://github.com/pandegroup/openmm (Accessed March 2018). ().
60. Monroe Jacob I, Shirts Michael R. Converging free energies of binding in cucurbit[7]uril and octa-acid host-guest systems from SAMPL4 using expanded ensemble simulations. J Comput Aided Mol Des. 2014;28(4):401–15. https://doi.org/10.1007/s10822-014-9716-4.
61. Muddana HS, Fenley AT, Mobley DL, Gilson MK. The sampl4 host-guest blind prediction challenge: an overview. J Comput Aided Mol Des. 2014;28(4):305–17. https://doi.org/10.1007/s10822-014-9735-1.
62. [n. d.] Md trajectories of ala2. https://figshare.com/articles/new_fileset/1026131 (accessed March 2018). ().
63. Wang F, Landau DP. Efficient, multiple-range random walk algorithm to calculate density of states. Phys Rev Lett. 2001;86:2050–3.
64. Shirts MR, Chodera JD. Statistically optimal analysis of samples from multiple equilibrium states. J Chem Phys. 2008;129:124105.
65. Tiwary P, Berne BJ. Spectral gap optimization of order parameters for sampling complex molecular systems. Proc Natl Acad Sci. 2016;. https://doi.org/10.1073/pnas.1600917113 eprint: http://www.pnas.org/content/early/2016/02/24/1600917113.full.pdf.