# Ensemble Toolkit: Scalable and Flexible Execution of Ensembles of Tasks

Vivekanandan Balasubramanian, Antons Treikalis, Ole Weidner*, Shantenu Jha†
Department of Electrical and Computer Engineering, Rutgers University, Piscataway, New Jersey 08854
*Work done when at Rutgers
†Corresponding Author



***Abstract*—There are many science applications that require scalable task-level parallelism, support for flexible execution and coupling of ensembles of simulations. Most high-performance system software and middleware, however, are designed to support the execution and optimization of single tasks. Motivated by the missing capabilities of these computing systems and the increasing importance of task-level parallelism, we introduce the Ensemble toolkit which has the following application development features: (i) abstractions that enable the expression of ensembles as primary entities, and (ii) support for ensemble-based execution patterns that capture the majority of application scenarios. Ensemble toolkit uses a scalable pilot-based runtime system that decouples workload execution and resource management details from the expression of the application, and enables the efficient and dynamic execution of ensembles on heterogeneous computing resources. We investigate three execution patterns and characterize the scalability and overhead of Ensemble toolkit for these patterns. We investigate scaling properties for up to O(1000) concurrent ensembles and O(1000) cores and find linear weak and strong scaling behaviour.**

## I. Introduction

Many scientific applications in the field of molecular sciences, computational biology [1]–[3], astrophysics [4], weather forecasting [5], [6], bioinformatics [7] are increasingly reliant on the ability to run ensemble-based methods to make scientific progress. This is true for applications that are both net producers of data, as well as aggregate consumers of data. Computationally, an ensemble is comprised of multiple concurrent units of execution, which are referred to as *tasks*. Tasks may refer to simulations, analysis or any independent computational process that needs to be executed. Different ensemble-based application vary in the degree of coupling between tasks, dependency between stages of tasks, as well as the level of heterogeneity across tasks.

In spite of the apparent simplicity of running ensemble-based applications, the scalable and flexible execution of a large and collective set of tasks is non-trivial. The requirements of ensemble-based applications are more complex than simple parameter sweeps: they require varying degrees of coupling between the ensemble members, such as global synchronization, temporally varying pairwise interactions. In addition, all ensemble members must successfully complete as the collective properties of the entire ensemble are often computed.

There are many functional, performance and usability challenges that currently prevent ensemble-based applications from exploiting the full power of high-performance computing (HPC) systems. The situation is exacerbated by the fact that HPC software eco-system has traditionally been designed and optimized with large, monolithic applications. There is an absence of tools designed and implemented with ensembles as the fundamental unit. Hitherto, ensemble-based applications have either been retrofitted into one of many *general purpose* workflow systems, or wrapped in specifically constructed scripts that address some, but rarely all of the requirements. The former can in principle support ensemble-based applications as a *special* case, but most workflow systems have their design and performance tuned to managing complex inter-dependencies of tasks, which is not the dominant requirement of ensemble-based applications. The scripting-based approach presents a significant burden on the user, especially, when it comes to running large ensembles in a fault-tolerant way on heterogeneous resources or in developing sophisticated ensemble-based applications.

In response to these challenges and the growing importance of ensemble-based applications, we design and implement Ensemble toolkit with the following features to meet the requirements of ensemble-based applications: (i) abstractions for the expression of an ensemble of tasks, (ii) support for ensemble-based execution patterns, (iii) decoupling the execution and management of these patterns from their expression and, (iv) a runtime system for efficient execution of tasks and provides flexible resource utilization capabilities over a range of HPC platforms.

The design and implementation of Ensemble toolkit advances both systems and applications. While being novel in design and concept, its implementation builds upon well-established abstractions and significant reuse of existing software components. Although the original motivation of Ensemble toolkit arose from the needs of the bio-molecular science [8] and geoscience applications, the concepts, components and use of Ensemble toolkit is agnostic to the application domain. Ensemble toolkit fills a missing gap in the repertoire of production-grade HPC tools.

After a brief discussion in Section 2 of related tools and research, in Section 3, we present the objectives, design, and implementation of Ensemble toolkit. In Section 4, we perform validation and scalability experiments using different workloads. We conclude in Section 5 with a discussion of the key features and aspects of Ensemble toolkit.

## II. Related Work

Many tools have been developed to run workloads on HPC resources, however to the best of our knowledge, there are no best practices for ensemble-based applications, nor are there tools that embody those practises. For example, replica-exchange and other ensemble-based enhanced sampling algorithms in molecular science applications [1], [2], [9] use approaches that depend on the number of tasks, coupling and size of the system being studied. Similarly, in geosciences, ensemble-based simulations are used to study dispersion of atmospheric pollutants, effects of contaminant release [10], hurricane prediction [11]; existing solutions handle the coordination, orchestration and execution of ensembles differently.

In addition to the parameter sweep tools (e.g., Nimrod [12]) and scripting methods discussed earlier, multiple workflow systems [13] have been designed for HPC systems. Typically these are geared towards exploiting concurrency across complex task dependencies and typically use custom languages that increase the effort from the user-end. Systems such as Pegasus [14] and DAGMan [15] convert given workloads into directed acyclic graphs (DAGs). DAGMan simply schedules the jobs as per the DAG where each edge of the DAG specifies the order of precedence and primarily meant for static workloads. Pegasus , primarily expressed in virtual data language, maps workflows onto select execution sites by creating resource specific DAG. Workflow systems such as Swift [16] do not rely on the creation of a DAG but require an explicit mention of the order of execution of the various tasks in a multi-stage application.

There are other dataflow oriented tools such as Copernicus [17], Ruffus [18], Snakemake [19] and COSMOS [20]. Both Ruffus and COSMOS provide a python library to construct the user application whereas Snakemake provides a separate workflow definition language. Snakemake and COSMOS convert their workload into DAGs using tool dependencies and file naming conventions and enable execution on common HPC. Ruffus mainly concentrates on expression & automation of conventional pipeline based workloads.

To the best of our knowledge, few, if any workflow systems have been designed for ensemble-based applications and their execution patterns from first principles. Tigres [21], a recent library, for workflow composition provides "programming templates" to compose computational or data pipelines. By combining these templates it is possible to compose many ensemble-based applications. YAWL [22] provides a workflow language that enables application composition by supporting a list of patterns that were observed to be common in workflows.

Some tools can be used for ensemble-based applications with additional customization, but the extent and level at which additional customization is required varies across tools. One of the motivations of this paper is to design components, using a general-purpose language, to provide domain-independent functionality and thus minimize this "last mile" effort.

## III. Ensemble Toolkit

We provide a semi-informal discussion of the requirements of ensemble-based applications and methods to motivate the design and implementation of Ensemble toolkit.

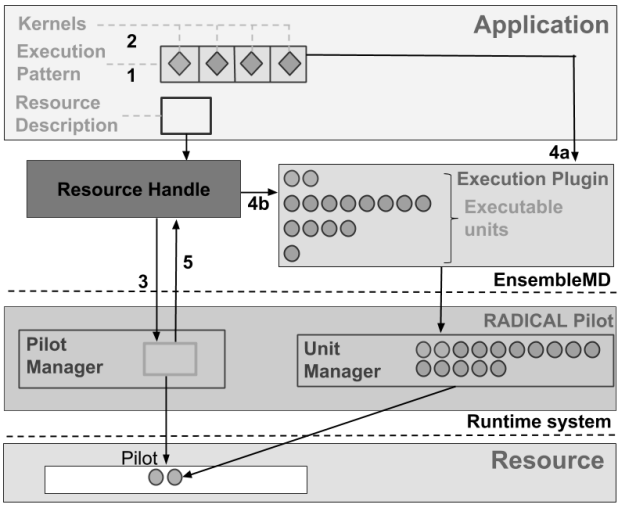


Fig. 1. Ensemble toolkit components: Execution Patterns, Execution Plugins, Resource Handle, Kernel plugins. Five steps: (1) Pick execution pattern for application, (2) Define Kernel plugins in the various stages of the pattern, (3) Create resource handle and submit a request for resources, (4) Execution plugin (a) binds execution pattern and the kernel plugins & (b) executes them on the remote machine, (5) User gets back control.

### A. Requirements

Ensemble-based applications vary in the type of coupling between tasks, the frequency and volume of exchange between these tasks, and the executable software ("kernel") of each ensemble member. Applications developers require simple and uniform approaches for developing diverse ensemble-based applications without compromising scale and the ability to use heterogeneous resources.

From a systems perspective, there is a need to provide efficient and flexible resource management over a diverse range of resource types. Often the total resource requirements of the ensemble of tasks are much greater than the resources instantaneously available. This can be due to the lack of resources, long queue wait times or upper limits enforced by scheduling policies. In either case, it is important to decouple the total resources required, from the resources utilized (or available) at any instant of time. This is not easily available on HPC systems and points to the need for a special-purpose runtime system.

### B. Design

Three primary design objectives of Ensemble toolkit are:

1) Support the easy development of a range of ensemble-based execution patterns and applications
2) Manage execution details while providing performance
3) Support dynamic resource management so as to decouple execution details from the instantaneous availability of resources.

Analyzing many ensemble-based applications we observe few common and recurring ways in which ensembles communicate and are synchronized. We take advantage of this recurrence by creating abstract "execution patterns" which are agnostic to the type of workload, the number of tasks and resource requirements of tasks. For example, an execution

pattern of a *bag of tasks* would create a set of tasks that are independent of each other without specifying the actual work done or their resource requirements.

The execution pattern is a core component of the Ensemble toolkit. The Ensemble toolkit provides a mechanism to develop applications by parametrizing pre-defined patterns and defining its execution stage(s). We identify and support unit patterns which are a set of atomic, unique execution patterns that capture different, exclusive modes of communication and synchronization patterns. These unit patterns can then be combined to form higher-order patterns consisting of more complex communications and synchronizations. A user is required to provide only the application logic workload via the exposed components. The details of task creation, submission and synchronization, data management, etc., are hidden in the lower layers of Ensemble toolkit while resource management, task execution, data movement, etc are performed by the underlying runtime system that Ensemble toolkit utilizes.

The resulting architecture of Ensemble toolkit (Fig. 1) has four components: execution patterns, resource handle, kernel plugins and execution plugins. These components ease application development while confining the execution complexity to the runtime system. We discuss each component:

1) **Execution Pattern:** It is a high-level object that represents the synchronization and communication patterns of ensembles. They can be viewed as parametrized templates that capture the execution of the ensemble(s).
2) **Kernel plugin**: It is an object that abstracts a computational task in Ensemble toolkit. It represents an instantiation of a specific science tool along with the required software environment. Kernel plugin hides kernel-specific peculiarities across different resources as well as differences between their interfaces
3) **Resource Handle**: It provides methods to allocate, run an execution pattern and de-allocate resources.
4) **Execution Plugin**: The execution plugin binds the kernel plugins and the execution pattern, and translates the tasks into executable units. Along with resource details from the resource handle, these executable units are forwarded to the underlying runtime system, thus decoupling execution from the expression of the application.

The resource handle, execution pattern and kernel plugins are user facing components used to obtain details regarding the resources, pattern used and workload at each stage of the pattern. The execution plugin is an internal component of the toolkit which is responsible for receiving information from other components and passing to the runtime system. It currently supports static binding and translation. "Intelligence" can be added so as to support applications with specific workload or resource requirements. To do so requires the use of information about the resource(s) coupled with requirements from the application, to devise execution strategies [23]

### C. Implementation

In order to keep the software footprint of Ensemble toolkit small, existing tools, libraries and frameworks were used wherever possible. To be able to use multiple HPC systems, Ensemble toolkit follows a standard job submission language.

*1) Job submission language:* The Ensemble toolkit is designed to be use the SAGA API [24] which is consistent with the standard Job Submission Description Language [25].

*2) Runtime system:* Ensemble toolkit relies on a runtime system to manage task execution, data movement and resource management. One of the design objectives was to provide dynamic resource management, a type of which is to be able to execute more tasks than available resources would allow. Ensemble toolkit achieves this via the use of Pilot-Job systems [26]. Pilot-Job systems provide placeholders or container jobs that are submitted to the target resource. These container jobs enable application-level scheduling of any number of workload tasks on these resources. Of the many pilot-systems currently developed [26], we select RADICAL-Pilot (RP) [27]. This is due to certain features unique to RADICAL-Pilot: 1) Support for MPI-based applications, 2) well-characterized performance [27] and, 3) designed to support execution on heterogeneous resources due to its architecture and use of SAGA. It is important to note that the Ensemble toolkit can be coupled with other general-purpose or special-purpose runtime systems. Ensemble toolkit is actively developed and provides the necessary hooks for this coupling.

### D. Ensemble toolkit Execution Patterns

The Ensemble toolkit supports three execution patterns: ensemble of pipelines, ensemble exchange (EE) and simulation analysis loop (SAL). These patterns were motivated by their frequent use in the molecular and geo-sciences. These patterns can be used for any application that follows the same orchestration of tasks by modifying the specific execution kernel plugins. We take a look at these patterns in terms of the coupling and dependencies of tasks.

*1) Ensemble of Pipelines:* It consists of an ensemble of independent pipelines of tasks. Each pipeline consists of multiple stages representing a well-defined execution order; each stage can contain heterogeneous workloads. Although each stage of a pipeline depends on its predecessor, the pipelines execute independent of each other. Figure 2(a) is a pictorial representation of the ensemble of pipelines pattern consisting of `N` pipelines each with `M` stages.

*2) Ensemble Exchange (EE):* The EE pattern consists of interacting ensemble members, where an ensemble member can be in one of two possible states. An ensemble member may interact with other ensemble members while in the same stage. There is no obligatory global barrier (synchronization) across ensemble members. As depicted in Figure 2(b), each ensemble member executes independently but interacts with other members in the exchange stage. A common example of the EE pattern is the replica exchange molecular dynamics (REMD) algorithm where the exchange between the MD simulations is not temporally synchronized and is pairwise.

*3) Simulation Analysis Loop (SAL):* The SAL pattern is a two-stage iterative execution pattern. The first stage consists of an ensemble of simulation instances and second stage consists of an ensemble of analysis instances. Figure 2(c) represents the SAL pattern with N simulation instances followed by M analysis instances in each iteration. Simulation ensemble members are synchronized before transitioning, as are the analysis ensemble members.

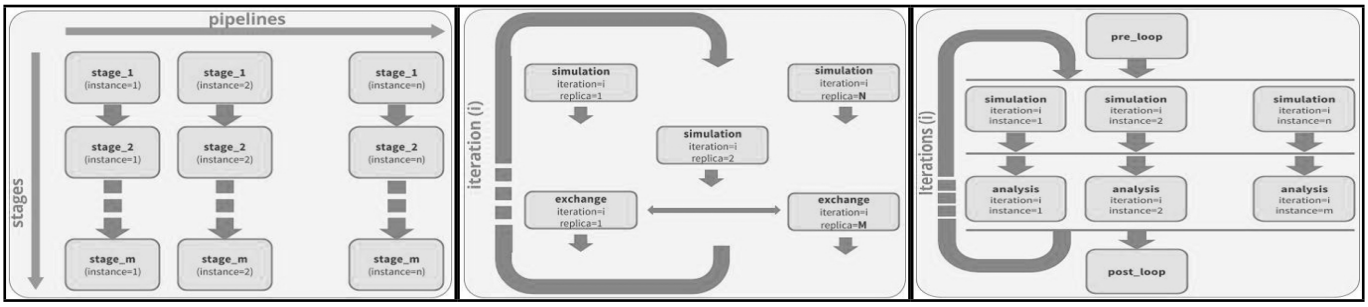


Fig. 2. Execution patterns supported by Ensemble toolkit: (a) Ensemble of Pipelines (b) Ensemble Exchange (c) Simulation Analysis Loop

## IV. Characterization and Validation

In this section, we characterize the performance of the toolkit and validate its design. We measure the the total time to completion (TTC) but decompose it into relevant primary components. We divide our experiments into two tracks: validation and performance characterization. The execution patterns are characterized by using the same workload for three patterns. Next, we examine the effect of switching kernels while the pattern is kept constant. For the second experiments track, we characterize the scalability of Ensemble toolkit– we run strong and weak scaling tests for the EE and SAL execution patterns.

The HPC machines used for the experiments are the same platforms which scientific applications that use Ensemble toolkit will use for production runs. In the first two experiments, we use the XSEDE Comet [28], an Intel Xeon cluster with 1984 nodes and 24 cores per node and 120GB memory per node. In the scaling experiments, we use XSEDE Stampede [28], an Intel Xeon cluster with 6400 nodes and 16 cores per node and 32GB memory per node and XSEDE Supermic [29], an Intel Xeon Phi cluster with 360 nodes with 20 cores per node and 60GB memory per node.

### A. Characterization of Execution Patterns

To validate three existing execution patterns in Ensemble toolkit– ensemble of pipelines, EE, and SAL, we create a two-stage application for each pattern. We use the `mkfile` kernel in the first stage to create a file in each task and the `ccount` kernel in the second stage to count the number of characters in each of those files.

In this experiment, we vary the number of tasks and cores from 24-192, keeping their ratio 1:1. We use the XSEDE allocated Comet cluster. A decomposition of the total time using the three different patterns is given in Fig 3. The first three subplots depict the execution time of the application using the three patterns. The following subplot presents the decomposition of the Ensemble toolkit overhead for the ensemble of pipelines pattern. As expected, we observed similar overheads when using EE and SAL execution patterns, hence we avoid presenting them again.

From the first three subplots, it can be observed that the application execution times remain relatively similar at all the configurations across patterns. This is due to the fact that each task has the same workload and all the tasks execute concurrently in all the patterns.

The Core overhead, which is the time taken to initialize Ensemble toolkit, launch and cancel resources requests, remains

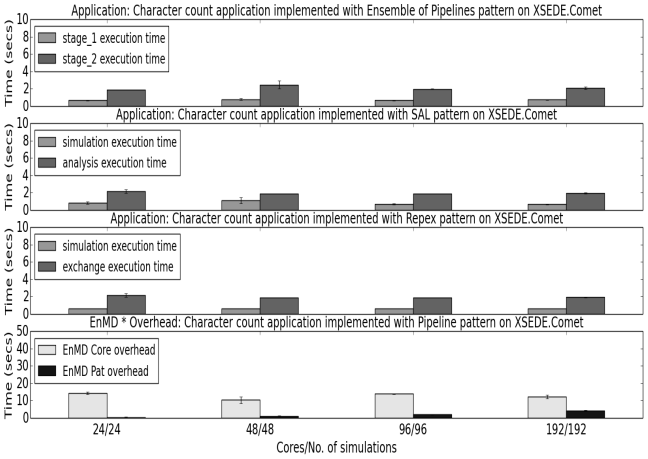


Fig. 3. Character count application implemented with pipeline, SAL and EE pattern on the XSEDE Comet. We vary, in same proportion, the number of tasks and cores from 24-192, thus all the tasks are concurrently executed. The first three subplots show the application execution times using the three patterns. The next subplot shows the Ensemble toolkit overhead.

constant in all the configurations as it is independent of the pattern or resource. The Ensemble toolkit Pattern overhead is the time taken by Ensemble toolkit in creating tasks using RADICAL-Pilot and submitting tasks for execution to RP. As expected it depends on the number of tasks. . There are overheads from RP (not shown) that arise from the time for task submission by RP on the remote machine, communication latencies, etc [27]. RADICAL-Pilot is being engineered to minimize these delays.

### B. Validation of Kernel Plugins

The objective of this experiment is to present and validate the support for kernel abstractions in Ensemble toolkit. We choose the SAL pattern and replace the kernel plugins with MD tools: Gromacs in the simulation stage and LSDMap [2] in the analysis stage. We use Comet and keep the data points in the same range (24-192).

Figure 4 presents a decomposition of the total time. We observe that, for the same range of tasks on the same machine, the overheads obtained with Gromacs and LSDMap are similar to the ones presented in Figure 3. It can be inferred that changing the kernel plugins and hence the workload does not effect the overhead presented by Ensemble toolkit.

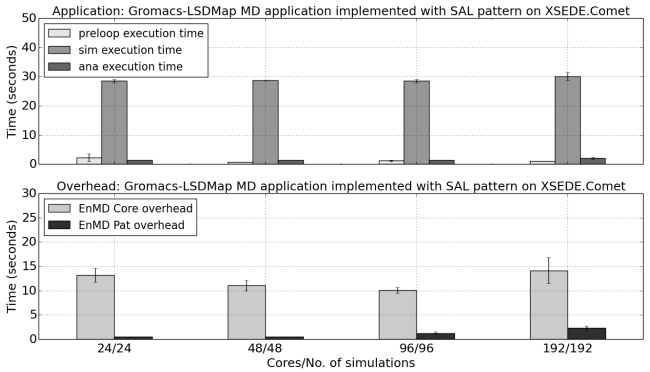


Fig. 4. Gromacs-LSDMap application implemented with SAL execution pattern on the Comet cluster in XSEDE. We vary the number of tasks from 24-192 but also vary the number of cores similarly.

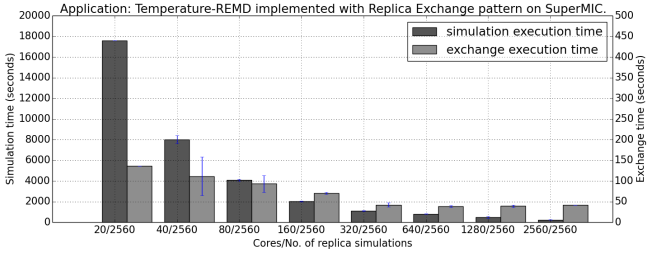


Fig. 5. Strong scaling test for EE execution pattern on XSEDE Supermic using Amber-Temperature Exchange kernel plugins and the alanine dipeptide molecule with 2560 replicas over a (20-2560) range of number of cores hence varying problem size per core.

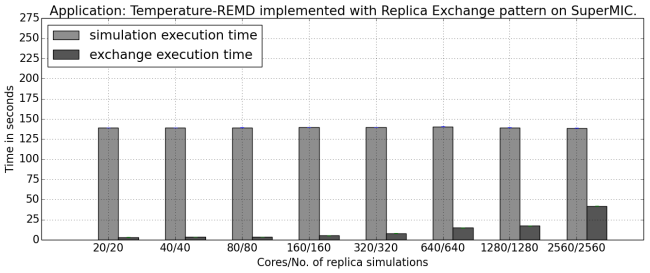


Fig. 6. Weak scaling test for EE execution pattern on XSEDE Supermic using Amber-Temperature Exchange kernel plugins and the alanine dipeptide molecule with fixed problem size per core.

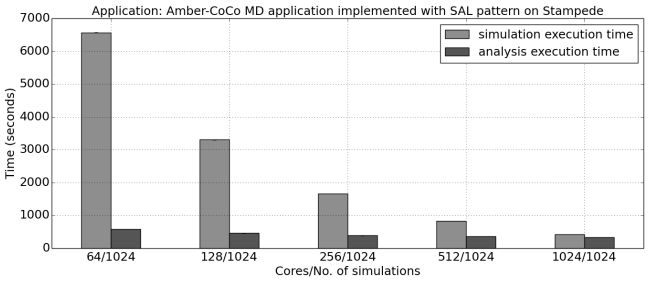


Fig. 7. Strong scaling test for Simulation-Analysis-Loop execution pattern on XSEDE Stampede using Amber-CoCo [1] kernel plugins and the alanine dipeptide molecule with 1024 simulations over a (64-1024) range of number of cores hence varying problem size per core.

### C. Characterization of Scalability

We have now validated both the support for different execution patterns and kernel plugins in Ensemble toolkit. We now test the scalability of the toolkit with real science workloads. We perform strong and weak scaling tests for EE pattern and the SAL pattern.

*1) Ensemble Exchange Pattern:* We run the EE pattern for a solvated alanine dipeptide molecule containing 2881 atoms. We use the Amber MD Engine for the simulations and perform a temperature exchange during the exchange stage. We perform both the experiments on the SuperMIC cluster in XSEDE [29]. Figure 5 and Figure 6 present the results of the strong and weak scaling experiments respectively.

For strong scaling experiments, we keep the number of replicas constant at 2560 and vary the number of cores between 20-2560. Each replica is run on 1 core for 6ps before exchange. In Figure 5, note that since the simulation and exchange times are different by orders of magnitude, we use two y-axes as labeled. From Figure 5, we can observe that the simulation time decreases to half its value when the number of cores are doubled. The exchange times, on the other hand, remain constant as they depend on the number of replicas, which is constant for this experiment.

In weak scaling experiments, we keep problem size per core constant, i.e., we keep the ratio of the number of replicas to the number of cores constant. We vary the number of replicas from 20-2560 and the number of cores proportionately. Each replica is run on 1 core for 6ps before the exchange. Our results in Figure 6 show that the simulation time remains relatively constant. The exchange times, however, increases as this depends on the number of replicas.

*2) Simulation Analysis Loop Pattern:* We now characterize the scalability of the toolkit by performing strong and weak scaling tests for the SAL pattern. We implement the iterative collective coordinates algorithm [1] using the SAL pattern. We use a solvated alanine dipeptide molecule containing 2881 atoms as our physical system. Each simulation is executed using the Amber MD Engine for 0.6 ps followed by the CoCo analysis of all simulations. Figure 7 and Figure 8 present the results of our strong and weak scaling experiments.

In the strong scaling experiment, we keep the number of simulations fixed at 1024 and use one core per simulation. We vary the number of cores used from 64-1024. We observe that the simulation time decreases linearly with increase in the number of cores. The analysis algorithm is executed in serial and thus depends on the number of simulations. Hence, the analysis execution time remains constant for all configurations.

In the weak scaling experiments, the number of cores is varied between 64-4096 while the number of simulations is kept equal to the number of cores at all times. For each of these configurations, the simulation execution time is observed to be constant. The analysis is executed in serial and thus increases with the number of simulations. Note that the absolute performance of the analysis kernel is unrelated to the scalability of Ensemble toolkit.

**Analysis of scaling results:** In the strong scaling experiments, as we increase the number of cores, the number of simulations executing in parallel increase. As the total number of simulations is greater than or equal to the number of cores for any configuration, the TTC decreases with increasing core

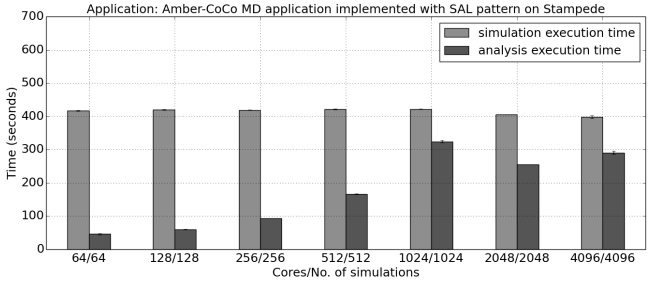


Fig. 8. Weak scaling test for Simulation-Analysis-Loop execution pattern on XSEDE Stampede using Amber-CoCo kernel plugins and the alanine dipeptide molecule with number of replicas equal to the number of cores hence fixed problem size per core.

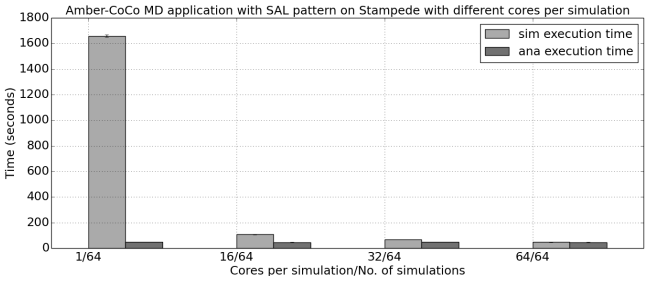


Fig. 9. Amber-CoCo MD application using the SAL execution pattern on XSEDE Stampede with varying no. of cores per simulation for a fixed no. of simulations

availability. In the weak scaling experiments, we keep the problem size per core constant and use as many cores as there are simulations. Thus, all simulations execute in parallel at all configurations and we observe almost constant execution time.

The observed linear behaviour suggests that scalability is invariant of the patterns. The linear scaling with the size of the ensemble, i.e., number of ensemble members, is a consequence of the capabilities of the runtime system. This behaviour attests to both an important feature of Ensemble toolkit and validates the choice of using RADICAL-Pilot as the runtime system.

It important to note that although we have used one core for each simulation in these experiments, running multi-core simulations will not change the scaling behaviour. This is due to the fact the overheads, both from Ensemble toolkit and RADICAL-Pilot, depend on the number of tasks as opposed to the size of each task. Whereas the size of each task might affect the absolute execution time, the behaviour across different configurations of cores and tasks would remain consistent with the results obtained for single-core simulations.

*3) MPI capability:* We briefly demonstrate the MPI support in Ensemble toolkit by using the same Amber-CoCo MD application, but now with more than one core per simulation. In Figure 9, we fix the number of concurrent simulations at 64, increase the duration of simulation ten-folds to 6ps and vary the number of cores per simulation as 1,16,32,64 (and hence the total no. of cores 64, 1024, 2048, 4096). We see that the execution time of the simulations drops linearly with the number of cores used. Along with the strong and weak scaling experiments, this attests to the fact that given access to the required amount of resources, O(1000) tasks of both MPI and non-MPI type can be supported by Ensemble toolkit.

## V. Discussion and Conclusion

The objective of Ensemble toolkit is to construct simple yet flexible approaches to compose ensemble-based workflows. In Section III we outlined its primary requirements, design and implementation. Execution patterns expose common coordination and synchronization patterns, while the user provides workload definition and resource information. We introduced three execution patterns that represent the majority of ensemble-based applications and serve as useful unit patterns from which to construct more complex patterns. Experiments in Section IV, validated the design of execution patterns and kernel plugins and characterized the performance of Ensemble toolkit as a function of ensemble size.

Scalability experiments for the EE and SAL pattern with real workload showed linear strong and weak scaling behavior for up to O(1000) tasks. As measured by overheads, Ensemble toolkit does not impose any significant or fundamental scalability limits; performance and scalability are essentially determined by the pilot-based runtime. RADICAL-Pilot has been engineered to support up to 8K tasks on XSEDE Stampede and 2K tasks on Cray machines [27], [30]. O(10,000) tasks are being tested currently on NSF Blue Waters machines [31]. A technical road-map exists for O(100,000) concurrent tasks. These runtime performance enhancements will be seamlessly applicable to Ensemble toolkit.

This paper provides two important lessons for the workflow community: First, that the "building blocks" approach to constructing "functionally-constrained" or "special-purpose" workflow systems as exemplified by Ensemble toolkit is a promising one. Second, special-purpose workflow systems are likely to be more useful for complex but structured workflows such as the bio-molecular applications that adhere to the patterns in this paper. In other words, special-purpose workflow systems such as Ensemble toolkit provide an effective middle ground between re-factoring applications to use generic but difficult workflow systems, and completely unstructured approaches such as scripting.

Ensemble toolkit is currently in prototype stage, but it supports several active ensemble-based applications as part of the ExTASY [8] project and replica-exchange applications [32]. These applications expose fewer degrees-of-freedom and customization options but are functionally complete and provide required performance. Ensemble toolkit thus provides some empirical data in the "schism" and debate between general-purpose but monolithic workflow system versus special-purpose workflows constructed using building blocks and abstraction-based approaches.

There are multiple enhancements to Ensemble toolkit that are planned or underway: One avenue of research is to identify a *complete set* of unit patterns that can be used to compose *any* higher order "complex" pattern. The ability to express these higher order patterns as functions of unit patterns will enhance the ability to support more applications.

Currently, workloads are adapted to resources that are cho-

sen based upon user choice and independent of the dynamic state of workloads. Ref. [23] formalized and introduced *execution strategies* as the time-ordered set of decisions needed to execute a workload on dynamic resources. This requires integrating both application-level and resource information, which will see the execution plug-in transition from being a simple translation layer to an intelligent middleware component. The transition from static workload-resource mapping to dynamic mapping will enable efficient and optimized execution capabilities. It will lead to the ability to efficiently select resources for a given workload, as well as form the basis to adapt workloads to optimally utilize a pre-specified set of resources.

Many applications are not pre-defined as the type and amount of workload may be determined during execution time. Ensemble toolkit will progressively support more adaptive scenarios, for example the ability to kill-replace tasks [33], vary the number of tasks between stages, vary the workload in each task during execution time. The complexities arise from the adaptive execution and workloads, as well as the challenge in providing user-facing components to convey the decision points, and thus the adaptivity, meaningfully. In general, Ensemble toolkit forms an initial prototype of a software system upon which to develop advanced adaptive simulation algorithms [34].

**Software and Data:** Ensemble toolkit can be found at http://radicalensemblemd.readthedocs.org/en/latest and is released under the MIT license. An updated version of this paper is available at http://arxiv.org/abs/1602.00678. Raw data and scripts to reproduce experiments can be found at https://github.com/radical-experiments/enmd-pattern-testing.

**Acknowledgements:** We thank members of the RADICAL group for significant discussion in the design, testing and documentation of Ensemble toolkit: Ming Tai Ha for his feedback and work on testing and documentation, Mark Santcroos and Andre Merzky for help with RADICAL-Pilot, Nikhil Shenoy for initial experiments. We also thank Iain Bethune (EPCC) for testing and feedback on software, and other members of the ExTASY project. We also thank Peter Kasson, Thomas Cheatham and Michael Shirts for useful discussion about adaptive execution patterns. We also thank Matthieu Lefebvre, Ryan Modrak and Jeroen Tromp for insight into ensemble applications in seismic tomography. This work was funded by NSF CHE-1265788 and NSF ACI 1440677.

## REFERENCES

[1] C. A. Laughton *et al.*, "Coco: a simple tool to enrich the representation of conformational variability in nmr structures," *Proteins: Structure, Function, and Bioinformatics*, vol. 75, no. 1, pp. 206–216, 2009.

[2] J. Preto and C. Clementi, "Fast recovery of free energy landscapes via diffusion-map-directed molecular dynamics," *Physical Chemistry Chemical Physics*, vol. 16, no. 36, pp. 19 181–19 191, 2014.

[3] T. E. Cheatham III and D. R. Roe, "The impact of heterogeneous computing on workflows for biomolecular simulation and analysis," *Computing in Science & Engineering*, vol. 17, no. 2, pp. 30–39, 2015.

[4] E. Sirko, "Initial conditions to cosmological n-body simulations, or, how to run an ensemble of simulations," *The Astrophysical Journal*, vol. 634, no. 2, p. 728, 2005.

[5] H. Wan *et al.*, "Short ensembles: an efficient method for discerning climate-relevant sensitivities in atmospheric general circulation models," *Geoscientific Model Development*, vol. 7, no. 5, pp. 1961–1977, 2014.

[6] P. Bauer *et al.*, "The quiet revolution of numerical weather prediction," *Nature*, vol. 525, no. 7567, pp. 47–55, 2015.

[7] J. Martin *et al.*, "Rnnotator: an automated de novo transcriptome assembly pipeline from stranded rna-seq reads," *BMC genomics*, vol. 11, no. 1, p. 663, 2010.

[8] V. Balasubramanian *et al.*, "Extasy: Scalable and flexible coupling of md simulations and advanced sampling techniques," 2016, (under review) https://arxiv.org/abs/1606.00093.

[9] Y. Sugita and Y. Okamoto, "Replica-exchange molecular dynamics method for protein folding," *Chemical physics letters*, vol. 314, no. 1, pp. 141–151, 1999.

[10] G. Cervone *et al.*, "Risk assessment of atmospheric emissions using machine learning," *Natural Hazards and Earth System Science*, vol. 8, no. 5, pp. 991–1000, 2008.

[11] T. M. Hamill *et al.*, "Future of ensemble-based hurricane forecast products," *Bull. Amer. Meteor. Soc*, 2010.

[12] D. Abramson *et al.*, "High performance parametric modeling with nimrod/g: killer application for the global grid?" in *Parallel and Distributed Processing Symposium, 2000. IPDPS 2000. Proceedings. 14th International*, 2000, pp. 520–528.

[13] I. J. Taylor *et al.*, *Workflows for e-Science: scientific workflows for grids*. Springer Publishing Company, Incorporated, 2014.

[14] E. Deelman *et al.*, "Pegasus: Mapping scientific workflows onto the grid," in *Grid Computing*. Springer, 2004, pp. 11–20.

[15] C. Team, "Dagman: A directed acyclic graph manager," *See website at http://www. cs. wisc. edu/condor/dagman*, 2005.

[16] M. Wilde *et al.*, "Swift: A language for distributed parallel scripting," *Parallel Computing*, vol. 37, no. 9, pp. 633–652, 2011.

[17] S. Pronk *et al.*, "Molecular simulation workflows as parallel algorithms: The execution engine of copernicus, a distributed high-performance computing platform," *Journal of Chemical Theory and Computation*, vol. 11, no. 6, pp. 2600–2608, 2015, pMID: 26575558. [Online]. Available: http://dx.doi.org/10.1021/acs.jctc.5b00234

[18] L. Goodstadt, "Ruffus: a lightweight python library for computational pipelines," *Bioinformatics*, vol. 26, no. 21, pp. 2778–2779, 2010.

[19] J. Köster and S. Rahmann, "Snakemakea scalable bioinformatics workflow engine," *Bioinformatics*, vol. 28, no. 19, pp. 2520–2522, 2012.

[20] E. Gafni *et al.*, "Cosmos: Python library for massively parallel workflows," *Bioinformatics*, p. btu385, 2014.

[21] V. Hendrix *et al.*, "Tigres workflow library: Supporting scientific pipelines on hpc systems," in *IEEE CCGrid*, 2016.

[22] W. M. Van Der Aalst *et al.*, "Yawl: yet another workflow language," *Information systems*, vol. 30, no. 4, pp. 245–275, 2005.

[23] M. Turilli *et al.*, "Integrating Abstractions to Enhance the Execution of Distributed Applications," in *Proceedings of 30th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2016, http://arxiv.org/abs/1504.04720.

[24] A. Merzky *et al.*, "SAGA: A standardized access layer to heterogeneous distributed computing infrastructure" *SoftwareX*, 2015, dOI: 10.1016/j.softx.2015.03.001. [Online]. Available: http://dx.doi.org/10.1016/j.softx.2015.03.001

[25] A. Anjomshoaa *et al.*, "Job submission description language (jsdl) specification, version 1.0," in *Open Grid Forum, GFD*, vol. 56, 2005.

[26] M. Turilli *et al.*, "A Comprehensive Perspective on Pilot-Jobs," 2016, (under review) http://arxiv.org/abs/1508.04180.

[27] A. Merzky *et al.*, "RADICAL-Pilot: Scalable Execution of Heterogeneous and Dynamic Workloads on Supercomputers," 2015, (under review) http://arxiv.org/abs/1512.08194.

[28] "Xsede," https://www.xsede.org/high-performance-computing (accessed January 2016).

[29] "Lsu supermic," https://portal.xsede.org/lsu-supermic (accessed January 2016).

[30] M. Santcroos *et al.*, "Executing Dynamic Heterogeneous Workloads on Crays with RADICAL-Pilot," in *CRAY User Group 2016*, 2016.

[31] NSF Award (PRAC), http://goo.gl/nWFsCj.

[32] A. Treikalis *et al.*, "Repex: A flexible framework for scalable replica exchange molecular dynamics simulations," in *Proceedings of the 45th International Conference on Parallel Processing (ICPP)*, 2016, http://arxiv.org/abs/1601.05439.

[33] NSF Award (RAPID), http://goo.gl/e8RZE5.

[34] Shantenu Jha and Peter M. Kasson, "High-level software frameworks to surmount the challenge of 100x scaling for biomolecul ar simulation science," White Paper submitted to NIH-NSF Request for Information (2015), http://dx.doi.org/10.5281/zenodo.44377.