Running Ensemble Workflows at Extreme Scale: Lessons Learned and Path Forward

Kshitij Mehta[*] 0000-0002-9714-9981
Ashley Cliff[‡†] 0000-0001-7809-5546
Frédéric Suter[*] 0000-0003-1902-1955
Angelica M. Walker[†] 0000-0003-4308-6302
Matthew Wolf[*] 0000-0002-8393-4436
Daniel Jacobson[*†] 0000-0002-9822-8251
Scott Klasky[*] 0000-0003-3559-5772

[*]*Oak Ridge National Laboratory*, Oak Ridge, TN, USA
[‡]*Lineberger Comprehensive Cancer Center, University of North Carolina*, Chapel Hill, NC, USA
[†]*Bredesen Center for Interdisciplinary Research and Graduate Education*, *University of Tennessee,* Knoxville, TN, USA

***Abstract*—The ever-increasing volumes of scientific data combined with sophisticated techniques for extracting information from them have led to the increasing popularity of ensemble workflows which are a collection of runs of individual workflows. A traditional approach followed by scientists to run ensembles is to rely on simple scripts to execute different runs and manage resources. This approach is not scalable and is error-prone, thereby motivating the development of workflow management systems that specialize in executing ensembles on HPC clusters. However, when the size of both the ensemble and the target system reach extreme scales, existing workflow management systems face new challenges that hamper their efficient execution.**

In this paper, we describe our experience scaling an ensemble workflow from the computational biology domain from the early design stages to the execution at extreme scale on Summit, a leadership class supercomputer at the Oak Ridge National Laboratory. We discuss challenges that arise when scaling ensembles to several million runs on thousands of HPC nodes. We identify challenges with composition of the ensemble itself, its execution at large scale, post-processing of the generated data, and scalability of the file system. Based on the experience acquired, we develop a generic vision of the capabilities and abstractions to add to existing workflow management systems to enable the execution of ensemble workflows at extreme scales. We believe that the understanding of these fundamental challenges will help application teams along with workflow system developers with designing the next generation of infrastructure for composing and executing extreme-scale ensemble workflows.

***Index Terms*—workflows, ensemble, HPC, extreme scale**

I. Introduction

In many scientific domains, the investigation of important research questions requires performing multiple complex operations on large amounts of data that may be produced by scientific instruments, numerical simulations, or resulting from activity logging. The rapid increase in both the complexity of such scientific workflows and the volume of data they process has made it intractable to manually manage the execution of the different compute tasks and data movements that compose these workflows. Old-fashioned scripts written by domain scientists thus have left room for a myriad of Workflow Management Systems (WMS) to support the onus of the orchestration and execution of multiple interdependent tasks on distributed systems, and let scientists focus on more important questions.

More recently, and thanks to the ever-increasing capacity of leadership class supercomputers and the democratization of scientific computing on cloud resources, the complexity of scientific workflows has reached a new level. It is now common to combine the execution of multiple instances of a traditional workflow into a so-called ensemble workflow. For instance, in Molecular Dynamics, the expensive simulation of a long trajectory can be replaced by the simultaneous execution of multiple short-range simulations [1], [2]. The advent of Artificial Intelligence (AI) techniques to propose fast surrogate models of complex numerical simulations also calls for the definition of ensemble workflows to explore vast parameter spaces.

Domain scientists could logically expect that the execution of large ensemble workflows on extreme scale supercomputers could be handled by the same WMS so they can keep focusing on science. However, scaling up to millions of tasks and thousands of powerful and complex High Performance Computing (HPC) nodes creates a whole new set of challenges. This also comes with an important question: do the scientists have to adapt their applications, or can the WMS be improved to break this new scalability barrier and enable new scientific discoveries?

In this paper, we retrace the different challenges we faced with scaling up an application in the computational biology domain from the execution of a few instances on a small scale cluster in a lab to running a large ensemble workflow on one of the fastest supercomputers in the world. At each of these steps, we:

- Identify the intrinsic limitations of the execution mechanism used at that step;
- Identify and analyze the causes of the technical challenges we faced when increasing the scale;
- Propose solutions to these challenges that led us to the next step;
- Discuss how the problems we faced, the lessons we learned, and the solutions we opted for can be generalized beyond the specific use case considered in this paper.

The rest of this paper is organized as follows. Section II gives some details about the application and workflow management system we used and states which of their specific features are found in a broader range of applications and systems. Section III describes how ensemble workflows are traditionally run by domain scientists, and discusses the benefits of leveraging a workflow management system to run ensemble workflows. In Section IV, we list the new challenges that only appear when the scales of both the ensemble workflow and the target HPC systems become very large. Section V provides a path forward for workflow management systems to better handle the execution of large-scale ensemble workflows on large-scale HPC systems. Section VI discusses the related work, and we provide concluding remarks in Section VII.

II. Background

The main objective of this paper is to provide practitioners who want to run ensemble workflows at large scale with valuable insight about performance pitfalls they will likely face. Such insight has been gathered through our own experience of scaling up a science application to be able to exploit an HPC system using a workflow system that we introduce in this section. However, we believe that the main features of the considered application, workflow management system, and HPC system are very common and correspond to the way a broad range of scientific problems are currently solved.

A. Ensemble Workflow: iRF-LOOP

The use of AI techniques to solve complex problems is gaining traction in various science domains. For instance, in the computational biology domain, Iterative Random Forests [3] (iRF) are used for the creation of Predictive Expression Networks on the order of 40,000 genes or more [4]. By iteratively creating weighted forests, iRF takes advantage of the Random Forests ability to produce feature importance and lead to more accurate models. In each iteration, the importance scores from the previous iteration are used to weight features for the current forest.

The Iterative Random Forest Leave One Out Prediction (iRF-LOOP) application is a multi-threaded application written in C++ that can extract explainable properties of datasets and be used on a multitude of datasets to produce all-to-all associations [4]. Using a matrix with n features and m samples, iRF-LOOP will treat each feature as the dependent variable, or Y vector, and create an iRF model with the remaining n-1 features as the independent variables, or the X matrix. Like iRF, iRF-LOOP can produce meaningful insights even in cases where n is much larger than m. In practice, the iRF-LOOP model is created by running a separate iRF instance for each dependent variable. The larger the set of features, the more iRF runs that are needed. The result of each individual iRF run is a vector of size n of the importance of each independent feature in predicting the Y vector. Each of these runs executes a pipeline of two tasks, i.e., training followed by prediction, reading input files that contain *feature* information, and generating output files that are later post-processed to extract information. The list of output files includes importance vector files and a separate file for recording model weights. Following the completion of the n individual iRF runs, the n importance vectors are normalized and concatenated into an n x n directional adjacency matrix, with values that can be viewed as edge weights between the features.

The execution of iRF-LOOP for a large set of features to treat thus corresponds to the ensemble workflow shown in Figure 1. The structure of this particular workflow, with a set of independent pipelines to execute, followed by the aggregation and post-processing of the generated results, is very typical of ensemble workflows. This initial design of the workflow, in which a large ensemble of training and prediction tasks is executed which generates several output files which are then post-processed, has a strong impact on the scalability of the workflow on large supercomputers. Scaling up this particular workflow and running it on large HPC systems requires addressing challenges that should benefit other ensemble workflows from other scientific domains that exhibit a similar structure.

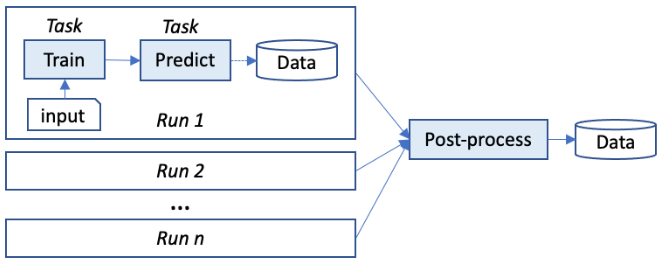

Fig. 1. The iRF-LOOP workflow ensemble. Each run of the ensemble executes a pipeline of two tasks - training and prediction. Each task produces output data that is later post-processed to extract information.

B. Workflow Management System: Cheetah

Cheetah is a workflow management system specifically designed for running large ensemble workflows on various systems [5]. Cheetah has been used successfully for studying topological parameters for in situ data analysis [6], investigating different data reduction methods for scientific data [7], developing a framework for whole device modeling simulations [8], and dynamically orchestrating applications coupled in memory [9].

Cheetah defines the *Campaign* abstraction to represent an ensemble of runs. Each run is a *pipeline*, i.e., a sequential or concurrent combination of compute tasks. As a simple example, a pre-processing step followed by a set of applications running in situ, followed by a post-processing application step forms a typical pipeline in many data-intensive science applications. Cheetah's Campaign abstraction can be used to setup a variety of execution patterns such as single pipelines, ensemble workflows, combination of serial and concurrently running tasks, and large-scale MPI workflows.

1) Campaign Composition: Similar to several modern workflow systems, Cheetah provides a Python-based interface to compose a Campaign. It provides the capability to create simulation runs and organize them into groups. Cheetah's composition interface allows setting up *runs* in an abstract way without having to write shell scripts to interact with the underlying system or its job scheduler.

To understand the outcome of composing and executing a very large campaign, it is important to know how Cheetah creates ensembles, and how it manages campaign metadata. Cheetah provides primitives to create pipelines and add applications to them. Users can express lists of values for different parameters across the application level, middleware layer, and system level. For each parameter value, a new *run* is created and added to the ensemble. As a simple example, a scientist may test 5 different compression algorithms, and further evaluate 5 different thread counts for each algorithm, to create a total of 25 runs for their campaign. Runs are then organized into groups called SweepGroups that correspond to batch jobs on clusters. Users provide the maximum wall time limit for each run and SweepGroup in the Campaign. Through its *machine* interface and native support for several supercomputer architectures, Cheetah can calculate the resources required to execute runs in the ensemble. The Cheetah object model is shown in Figure 2. Metadata is stored at all levels - Campaign, SweepGroups, and Runs to fully describe the ensemble and its components.

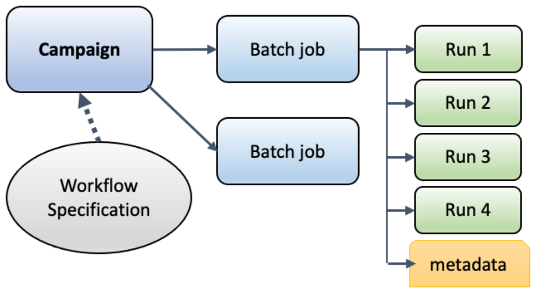

Fig. 2. The Cheetah campaign model. A *Campaign* is a hierarchy of *SweepGroups* or batch jobs, which contain the individual Runs of the ensemble with their own workspaces.

A Cheetah Campaign is mapped to a directory schema defined by Cheetah. SweepGroups are stored as separate directories, and ensemble Runs are provided their own workspace in the SweepGroup. Thus, runs in the ensemble are independent of each other, and several of them can execute concurrently depending on available resources. Each SweepGroup has a top-level JSON file that serves as the group's manifest. It includes all information needed to run the SweepGroup, which includes all runs, pipeline descriptions, application information, wall time limits, and resources requested for the SweepGroup.

Composing and executing an ensemble of runs is thus a two-step process. The composition step is used to set up the Campaign, its directory hierarchy, and the metadata on the target system. The execution step is used to launch the Campaign and execute its runs.

2) Campaign Execution: When a Campaign is launched, the Savanna runtime engine executes the ensemble workflow on the available resources. SweepGroups correspond to batch jobs on the HPC machine and are submitted to the underlying job scheduling system. Thus, submitting a full Campaign can submit several batch jobs to the system scheduler. Savanna monitors, executes, and reports on the status of the runs in the Campaign, and manages metadata to represent the output and state of runs. Partially completed Campaigns can be resumed by re-starting the Campaign, thereby providing support for fault-tolerant execution of ensembles. Savanna manages the compute resources on the user's behalf, and dynamically orchestrates runs from the ensemble as runs finish executing and resources become available. In the absence of such functionality, scientists have to manually write scripts to manage resources, which can lead to highly inefficient executions of ensemble workflows.

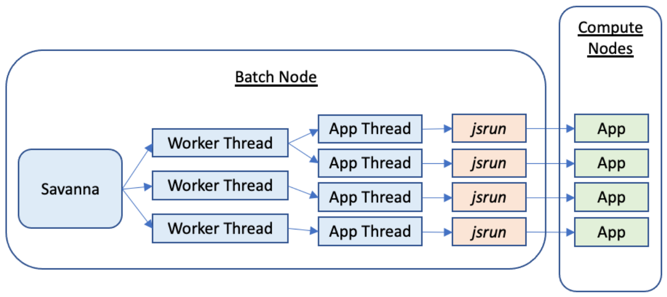

Fig. 3. Architecture of the Savanna execution engine. Savanna launches a hierarchy of threads for SweepGroups and ensemble runs. Savanna threads run on the cluster's batch node, whereas application processes run on the compute nodes. Metadata information is maintained in JSON files for a SweepGroup and individual runs in a SweepGroup.

Figure 3 shows the architecture of Savanna. It is a multi-threaded engine in which the master thread begins execution on the batch node of the cluster. As new runs in the ensemble get scheduled, Savanna spawns worker threads to manage the execution of runs and the tasks within. These threads monitor resource utilization and the status of currently executing ensemble runs. Tasks in a pipeline are executed on compute nodes via the underlying scheduler's jobstep command. Example of job step commands are *jsrun* for IBM's LSF scheduler [10] and *srun* for the Slurm scheduler [11].

3) Metadata Management: As with other WMS, Cheetah stores different types of metadata across various files. As shown in Figure 4, at the top level, every SweepGroup maintains two files: a read-only manifest of all runs in the group, and an execution status file created by Savanna for each run in the group. The manifest describes the ensemble - it describes the pipelines and their encompassing applications and tasks, and the list of all runs in the ensemble. The status file is a JSON document that Savanna uses to denote the execution status of all runs. During execution, Savanna stores the standard output and standard error for each task in a pipeline in separate files. It creates launcher scripts for launching every task in the pipeline. Furthermore, when a task completes, it records its return status and wallclock time in files.

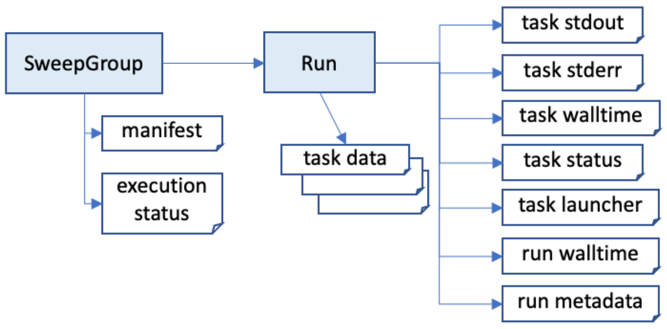

Fig. 4. Metadata management in the Cheetah ensemble workflow system. For every group of runs, a job manifest describes the runs in the job, and a status file denotes the execution status. Each run further creates several files for each task in the run.

HPC System: Summit

Summit is a leadership-class IBM system located at the Oak Ridge Leadership Computing Facility (OLCF). Summit is a pre-exascale machine with a theoretical peak double-precision performance of approximately 200 PetaFlops/s. Each of the approximately 4,600 compute nodes on Summit contains two IBM POWER9 processors with 21 user-level cores each, six NVIDIA Tesla V100 accelerators, 512 GB of DDR4 memory, 96 GB of High Bandwidth Memory (HBM2) for use by the accelerators, and 1.6TB of non-volatile memory. Each core supports 4-way hyperthreading so that an application can spawn a total of 168 threads on a node. Summit is connected to an IBM Spectrum Scale filesystem providing 250PB of storage capacity with a peak write speed of 2.5 TB/s.

As with many other supercomputing systems, Summit employs a three-level node design. Users get access to *login nodes* when they log in to Summit, where they can submit a batch job. When an allocation is granted to this job, it begins its execution on a *batch node*. The 'jsrun' job step command of the IBM's LSF scheduler is then used to execute tasks on *compute nodes*.

III. From a Traditional Approach to Using a Workflow Management System

In this section, we describe how ensemble workflows are traditionally run by domain scientists and the impacts this approach has on both their performance and scalability We then discuss the benefits of leveraging a workflow management system to run ensemble workflows and how it enables studies at larger scales.

Running Ensembles Using a Traditional Approach

The traditional way for a domain scientist to run an ensemble workflow, such as iRF-LOOP, on an HPC system is to rely on simple scripts rather than a more complex workflow management system. It typically requires a manual pre-processing phase in which the researcher creates a shell script for submitting jobs to the targeted compute cluster. This script interacts with the batch scheduler to request a set of nodes and run jobs on them. As the iRF-LOOP application generates several output files that later need to be post-processed, the user creates a hierarchy of directories on the file system to conveniently store the inputs and outputs of the individual runs of the ensemble. Then, they submit runs in groups or 'sets' to fully exploit the resources allocated through the batch scheduler. Upon successful completion of a set, the script submits the next set of runs on the available resources. This process is then repeated until all the runs in the ensemble have been executed. This approach is illustrated in Listing 1 for a small ensemble of iRF-LOOP runs.

```
#BSUB −nnodes 3

jsrun −p1 irfloop f1 &
jsrun −p1 irfloop f2 &
jsrun −p1 irfloop f3 &
wait

jsrun −p1 irfloop f6 &
jsrun −p1 irfloop f7 &
jsrun −p1 irfloop f8 &
wait
```

Listing 1. Shell script for running an IRF-LOOP ensemble. Each run requires one compute node. The number of concurrent job steps must match the number of nodes allocated.

Each run of the iRF-LOOP workflow runs on one compute node of Summit and uses all cpu cores available on the node. Three nodes are first requested from the batch scheduler to run the ensemble. This means that at most three runs can be launched concurrently. An explicit wait command acts as a barrier before the next set of three ensemble runs starts running.

This traditional, and mostly manual approach, has two main issues beyond putting all the burden of managing the ensemble workflow on the domain scientist, which is error-prone and not scalable. First, special care needs to be taken to ensure that only as many runs as the number of nodes allocated in the batch job are executed concurrently. Oversubscribing resources by submitting more runs than available compute nodes is likely to lead to a global execution failure. Second, this approach only works well if all the runs in the ensemble workflow have similar execution times. When different runs have widely varying runtimes, as is the case for iRF-LOOP and also for many applications whose execution time is input-sensitive, it leads to highly inefficient utilization of resources. This is illustrated by the top part of Figure 5. We can see the runs that finish early in their set cause the compute node to remain idle up to the next synchronization point. The overall throughput of the ensemble workflow execution essentially depends on the slowest run in each set of runs. Handling the straggler runs efficiently requires implementing advanced scheduling techniques that are usually not available to domain scientists who follow this script-based management approach.

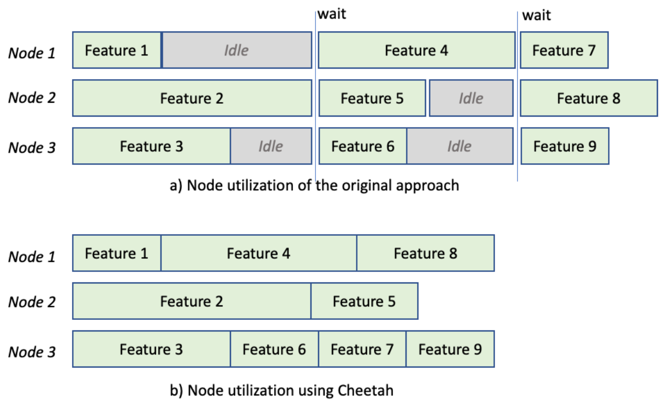

Fig. 5. Resource utilization in the original IRF-LOOP ensemble setup compared with the node utilization in Cheetah. The original method uses explicit wait statements to synchronize job steps, whereas Cheetah can dynamically track and provision resources at runtime.

Leveraging a Workflow Management System

To address the operational and performance issues raised by the traditional approach, we created a Cheetah specification for the iRF-LOOP ensemble. Domain scientists now only have to create the campaign specification in Python. They do not have to put a lot of thinking into the writing of shell scripts that fit the constraints of the underlying system nor to interact with the batch scheduler. The hierarchy of directories to store input and output files for each run is also handled by the workflow management system.

At runtime, this specification is automatically and transparently translated into launcher scripts and job submission commands. The WMS also dynamically provisions compute nodes for better resource utilization. Once a run is complete and its node is released, the workflow management system immediately schedules the next run on the free node. We can see in the bottom part of Figure 5 that compute nodes do not have to wait for an explicit synchronization anymore. This removes the idle times created in the traditional approach, hence improving the overall throughput.

Assessing the performance improvement

To further evaluate the performance benefits of running an ensemble workflow using a specialized workflow management system, we created an ensemble using the 2019 American Community Survey produced by the United States Census Bureau [12]. The dataset consists of 1,606 features for 3,220 counties obtained using the `tidycensus` package [13].

To run this full dataset using the traditional approach, multiple jobs were ultimately submitted to the batch scheduler, each with its own specific resource allocation request. Thus, multiple hand-written shell scripts have to be written and executed by the domain scientists.

Cheetah allows scientists to create the ensemble through its Python-based composition interface and easily express the parametric evaluation of the features of the dataset. Once the parameters to be explored were listed, they were grouped into a single SweepGroup that has all the runs of the full ensemble. The size of the resource allocation was then set for this SweepGroup but can be modified between executions. A partially complete ensemble can simply be re-submitted by modifying the resource allocation if desired. The metadata in the WMS are used to resume a partially completed ensemble from a previous checkpoint.

We measure performance in terms of the feature throughput per node-hour, which is the number of features executed in one node hour on Summit. Figure 6 compares the performance of the original approach with that of Cheetah. Using the former, we obtain a combined throughput of 0.8 features per node-hour for the full ensemble, whereas we obtain 7 features per node-hour with the latter using Cheetah, which results in an $8\times$ improvement over the original method.

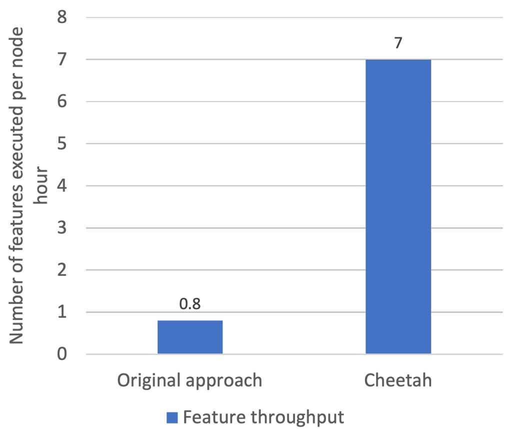

Fig. 6. Performance improvement obtained through using a workflow management system such as Cheetah. Feature throughput per node-hour on Summit improves from 0.8 features/node-hour using the conventional approach to 7 features/node-hour using the Cheetah WMS.

In addition to the performance advantages of using a WMS for executing ensembles, an abstract campaign specification further provides a portable way of running ensembles on different systems. The conventional approach requires creating submission scripts for every system on which iRF-LOOP is run. Using Cheetah, the same Python specification can be used on different systems with minimal changes.

Opening Way to Large-Scale Ensemble Studies

An additional benefit of using a workflow management system comes with the simplification of the expression of large-scale ensembles that were far beyond the reach following the traditional approach. The complexity of the scripts and the likeness of introducing errors or making bad scheduling decisions severely limit the scale of the imaginable studies.

In the specific case of running the iRF-LOOP ensemble workflow, using Cheetah as a workflow management system led the domain scientists to consider the processing of a dataset with over 81,000 features, where each feature consists of 50 unique train and test sets. This corresponds to an ensemble workflow that consists of over $81,000 \times 50$ = over 4,000,000 individual runs. From a performance scalability analysis standpoint, two campaign designs were considered:

- **Capability-class campaign design** - Create a single batch job containing all runs. This design allows us to explore running a large number of concurrent instances in the ensemble in a single job, whose resource allocation can be set to several thousand compute nodes. Capability-class jobs are designed to conduct large, wide runs in shorter amounts of time.
- **Capacity-class campaign design** - Create one batch job for each of the 81,000+ features in the ensemble with 50 runs for each feature in the job. Thus, the campaign consists of a large number of batch jobs that are submitted to the system. The system scheduler determines how the jobs are scheduled according to implicit policies. As each job consists of 50 runs that run on one compute node each, a batch job can scale up to a maximum of 50 nodes. A directory hierarchy in which every feature has its separate directory yields a more natural campaign design. Capacity-class designs may take longer to execute as the system scheduler can implement different priorities when faced with a large number of small-sized batch jobs.

Increasing the scale of an ensemble workflow to millions of individual tasks and targeting an execution at the full scale of a flagship supercomputer with thousands of nodes push the workflow management system to new limits and challenges that can only arise at such scales. In the next section, we list these challenges and identify their causes.

IV. Challenges in Ensemble Workflow Scaling

In this section, we describe the challenges encountered during the composition and execution of large ensemble workflows consisting of millions of individual runs. As shown in Figure 7, these challenges include memory management and performance of campaign creation, limitations of job queuing policies, task scaling, file system scaling, and post-processing of large amounts of data.

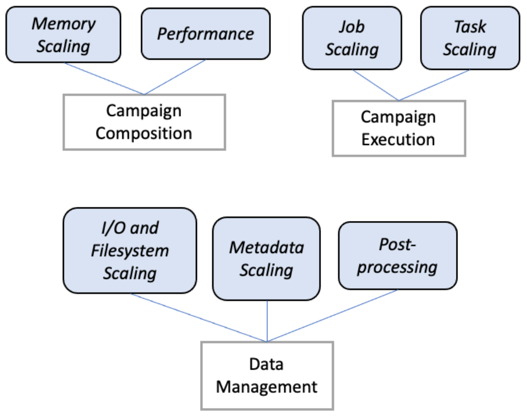

Fig. 7. A schematic of the challenges in scaling ensemble workflows to large scale. Challenges span the areas of campaign composition, execution, and overall data management.

A. *Challenges with Campaign Composition*

The Python-based Campaign specification interface of Cheetah provides abstractions to compose batch jobs and set independent workspaces for the runs composing the ensemble workflow. It allows scientists to represent applications, their runtime configuration, parameters, and environment settings.

1) Memory Management: The Cheetah composition function reads an ensemble specification created by the user and creates the `Campaign` object in memory along with the `SweepGroup` and `Run` objects. It also creates the campaign directory hierarchy on the system. For the capability-class design with a single large batch job consisting of millions of runs, the composition function ran out of memory to create all Run objects of the single large SweepGroup. As a workaround, the composition function's programming logic was modified to first create an empty SweepGroup and incrementally add Runs to it. Python iterator methods were used wherever applicable to read information about the runs sequentially and free up internal memory. This avoids creating the entire Campaign model in memory before writing it to the file system.

2) Performance: Composing a full ensemble workflow with several million runs took almost 4 hours. This includes reading the Python specification and creating the campaign directory hierarchy on the file system. To get a deeper understanding of the performance of creating a large ensemble, we profiled the composition function for an ensemble with 500,000 runs. Figure 8 shows the time taken by the different components of the composition function.

We observe that at large scale, file creation denoted as 'io open' takes 22% of the total time. The total I/O overhead including file creation and metadata operations takes almost 40% of the total time. File system and metadata operations include directory creation, running 'stat' operations on directories, and creating symbolic links to input data. JSON serialization and deserialization operations take 20% of the total time. As Cheetah maintains metadata in JSON files for each Run, and a JSON file that serves as a global manifest of Runs in a SweepGroup, JSON functions are called for each of

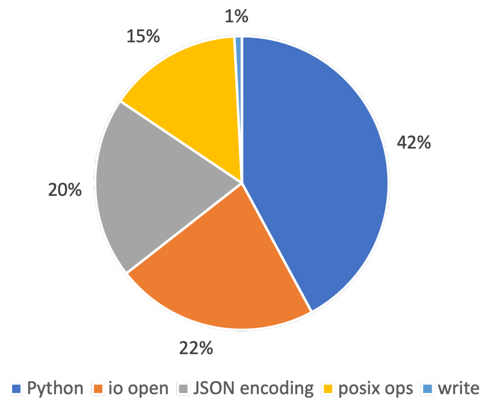

Fig. 8. Breakdown of the campaign creation time. The major components are Python computations, file system and I/O operations, and JSON serialization and deserialization operations. File system, I/O, and metadata operations take almost 40% of the time, and JSON operations take 20% of the total time.

the four million runs of the ensemble. This results in the high overhead of JSON serialization and deserialization operations. The difference in file system overhead for the capability- and capacity-class ensemble designs is negligible, as we have over 4,00,000 Run directories in a single SweepGroup for the capability-class ensemble, and over 4,081,000 directories for the capacity-class ensemble that contains one directory for each SweepGroup, which is an increase in directory count by 2%.

Although campaign creation is a one-time setup, a multi-threaded version of the composition function was developed to reduce the runtime for creating the large campaign. A simple data parallelism algorithm was used to distribute SweepGroups amongst threads. Figure 9 shows the performance improvements obtained by using multiple threads. These results were generated by composing an ensemble with 30,000 runs on a compute node on Summit.

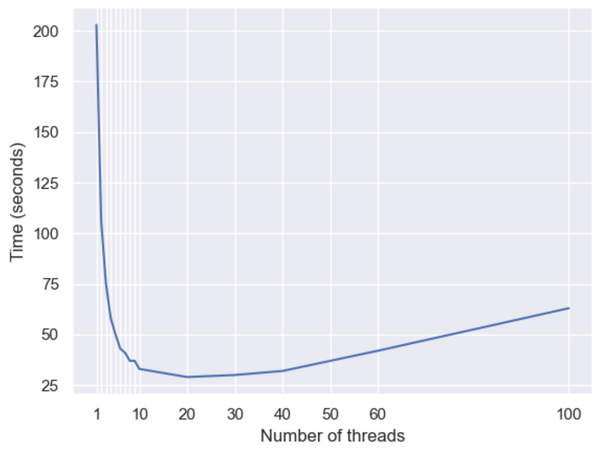

Fig. 9. Scaling the campaign creation operation using multiple threads on a compute node with 160 hyperthreads. The configuration with 20 threads shows the best performance.

We obtain the best performance using 20 threads which leads to 7× speedup over the single-threaded case. The multi-threaded composition function with 20 worker threads was then used to generate the large campaign with 4 million runs. The runtime was reduced to 85 minutes from 4 hours, which is a 2.7× improvement over the sequential case. While further analysis of the overhead is beyond the scope of our study, we estimate file system operations to take longer as the file and directory counts are much higher.

Lesson 1: Creating an ensemble and its directory structure can be memory-intensive and time-consuming at large scale.

B. Challenges with Campaign Execution

We now discuss the challenges associated with running the large campaign. Note that the capability-class design submits a single large job, whereas the capacity-class design submits a large number of jobs to the batch scheduler. The challenges with scaling a large ensemble include job scaling limitations and limitations with scaling individual jobs.

Job Scaling. The capacity-class ensemble design creates a SweepGroup each for the 80,000+ features of the iRF-LOOP ensemble. Submitting the full Cheetah campaign submits all the SweepGroups as individual batch jobs. However, job queue policies on supercomputers often limit the number of jobs that can be in a queue at a time for every user. The Perlmutter supercomputer at Lawrence Berkeley National Laboratory limits the number of jobs in the system to 5,000 [14], whereas the default job queue on Summit has a limit of 100 jobs. This prevents launching all SweepGroups of the ensemble together. To circumvent this limitation, several options were considered:

- Allow the WMS to submit jobs in batches to limit the number of concurrently running jobs. For instance, users would submit a configurable batch of 100 jobs at a time on Summit. However, this is only feasible for moderately-sized ensembles. For an ensemble with 80,000+ batch jobs, a user would have to manually submit a total of 800 jobs over the life of the full ensemble.
- Add a special post-processing task in every job that would submit the next batch job before the current job exited. However, addressing job timeouts and modifying a post-processing task for a running batch of jobs can become tedious. We discuss alternative solutions that provide deeper integration with job queue policies in Section V.

Lesson 2: Job queue policies restrict the number of concurrent batch jobs that can be run.

Task Scaling. As commands such as *jsrun* are submitted from a batch node, the number of concurrent runs is limited by the per-user process limit of 4,096 processes on a batch node. As each jsrun job step creates 3 processes, and JSM management may approximately create up to 23 processes, this creates an upper limit of approximately 1,350 simultaneous runs in a single batch node.

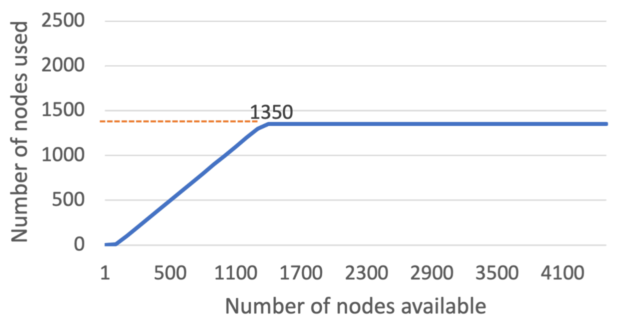

Fig. 10. The scalability limitation for the computational biology workflow in terms of the number of compute nodes allocated versus the number of compute nodes that can be used. A limit on the number of job steps that can execute concurrently limits the number of compute nodes that can be used.

As shown in Figure 10 this limits the scalability of a batch job in the capability-class campaign design to 1,350 nodes out of the approximately 4,600 nodes of Summit, which is only 30% of the full supercomputer capacity. Furthermore, for ensemble workflows that execute at a higher granularity of a task per CPU *core* as opposed to a task running on an entire node, the maximum number of concurrent runs is further limited as we need 42 calls to jsrun to use all 42 cores on a Summit node. This limits the scale to only $(1350/42) = 32$ nodes on Summit.

Lesson 3: The limit on the number of concurrent job step invocations limits the task scalability of the ensemble.

C. Challenges with Data Management

Metadata Scaling. WMS often use widely used file formats such as JSON for metadata storage as it is an easy-to-use, interchangeable, human-readable data format. Cheetah maintains the execution status of runs of a SweepGroup in a JSON file. During execution, the state of every ensemble run is managed and updated in this status file. However, updating a value in a JSON file causes the entire file to be loaded in memory every time before it is written back to the file system. For the capability-class ensemble design with a single metadata file, the metadata can become significantly large, e.g., over 500 Megabytes for our use case. As each run undergoes two status updates ('not_started' $\rightarrow$ 'running' $\rightarrow$ 'done'/'killed'), metadata updates quickly become a bottleneck for workflow scalability. Figure 11 shows the time taken to update a single field in a JSON file as its size increases.

At full scale with over 4 million entries, it takes 70 seconds to update a single field. Furthermore, special care needs to be taken to ensure that interruptions during an update do not leave the metadata file in an inconsistent state. While JSON has not traditionally been developed for large data, it remains popular for storing metadata due to its interchangeable format and hierarchical key-value pair schema.

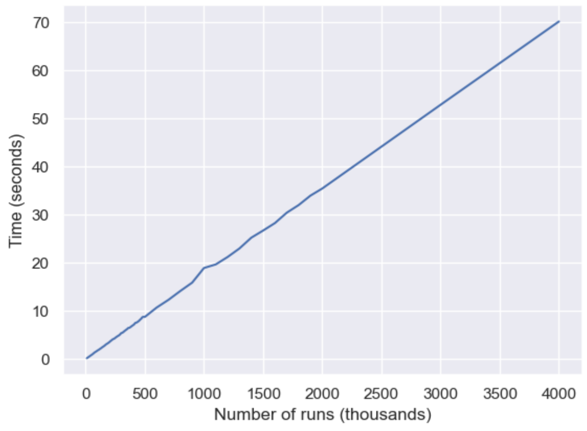

Fig. 11. Performance of metadata updates with the increasing scale of the ensemble. The X-axis shows the number of runs in the campaign manifest JSON file. The Y-axis shows the time to make a single update in the metadata file. The maximum value of 4 million runs represents the capability-class ensemble design. At the largest scale, it takes 70 seconds to update a single value in the JSON metadata file.

Lesson 4: Popular file formats for metadata management perform poorly at scale.

File System Scalability. The most challenging aspect of scaling ensemble workflows to the order of several million runs is the scalability of the file system. Table I lists the file system overhead for the large ensemble with iRF-LOOP.

TABLE I
FILESYSTEM OVERHEAD AT LARGE SCALE

No. of files created by iRF-LOOP per run	13
No. of files created by Cheetah per run of iRF-LOOP	12
No. of runs in the large ensemble	$>$ 4 million
No. of run directories for the full ensemble	$>$ 4 million
No. of files expected after full ensemble is completed	$>$ 100 million

Each run of the application workflow creates several files, whereas the WMS itself creates its own files for managing information. Thus, for 4 million runs in the campaign, over four million directories containing a total of over one hundred million files are created. This leads to prohibitively high overhead for the file system. The sheer number of files created can easily overwhelm the metadata server of the file system. We also run the risk of exceeding the number of *inodes* available per user on the supercomputer. As the file system is a shared resource and has limited bandwidth as compared to the compute capability of a machine, stressing it can have adverse effects on the overall performance of the workflow and the fidelity of data.

Lesson 5: Storing data in a few files per run of the ensemble easily leads to millions of files at scale.

1) Data Post-processing: Along with challenges associated with file system scalability, post-processing poses an additional challenge. Similar to many science workflows, iRF-LOOP performs a post-processing step in which *importance* vector files created in a Run are later analyzed to extract information. This step is performed after the entire ensemble is complete. Reading back several files from every Run ultimately requires reading back several million files, which poses a significant I/O bottleneck.

Lesson 6: Post-processing data from a large ensemble is prohibitively expensive.

V. Path Forward for Scalable Workflow Systems

The challenges identified for executing large ensembles span several areas of campaign execution and data management. Issues such as job and task scaling can be addressed by the WMS by providing more efficient orchestration mechanisms. However, file system issues arise due to the data model adopted by the application workflow and can be solved by modifying the application to use more efficient tools and libraries. However, avoiding code re-designs for specific use cases, and instead designing abstractions in the workflow management system is an attractive option for mitigating the gap between ensemble designs and their efficient execution at large scale. In this section, we discuss potential workarounds for the challenges described with running large ensemble campaigns on HPC machines.

A. Flexible Ensemble Orchestration

WMS such as HTCondor [15] can dynamically generate and schedule tasks, but there is need for a mechanism that allows concurrently spawning and coordinating multiple instances of the WMS itself. This will allow individual runs of the ensemble to be dynamically assigned to a running batch job, which will allow for easy switching between capacity-class and capability-class execution patterns.

B. Scalable Task Scheduling

Pilot job systems have been developed primarily to address scalable task scheduling. Pilot WMS manage resources on behalf of the user by spawning lightweight threads on compute resources and coordinating them with a master process. However, in cases where each run of the ensemble itself is a distributed application that uses an HPC framework such as MPI [16] for multi-processing, using Pilot WMS can be non-trivial. Pilot WMS need to evolve to seamlessly integrate with MPI where each run of the ensemble is an MPI application. The main workflow can also be modified to use MPI for nested parallelism by spawning sub-communicators for each run of the ensemble. However, in addition to requiring users to modify the application source code, libraries such as MPI cannot be used easily on architectures such as cloud environments, which causes a hindrance in porting the workflow to different architectures. Alternatively, a WMS can bypass the job step submission provided by schedulers and use the underlying process management interface such as PMIX [17] provided on clusters.

C. Abstractions for Data Management

It is typical for applications to use files to store data. The file model provides an easy and clear separation between logical data entities and is used widely for managing data. However, it does not scale well as the number of files increases into the millions or even hundreds of thousands. For ensemble workflows in which a first ‘pass’ such as a training loop creates millions of files which are then post-processed separately, a different approach is needed to ensure scalability. In the HPC domain, scientific data management libraries and frameworks that provide abstractions for storing and streaming data on different endpoints are used for managing large data efficiently. While they have not been traditionally used to address large data generated by ensemble workflows, abstractions for using efficient data formats, data streaming, and using the storage hierarchy on modern supercomputers can be highly impactful for managing large data. Figure 12 shows the opportunities for providing efficient data management in workflow management systems from the HPC domain.

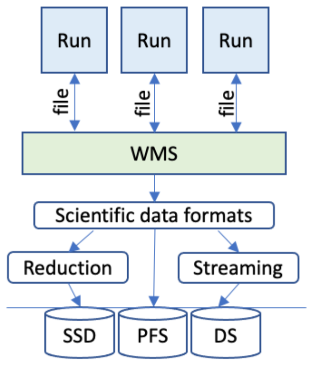

Fig. 12. Opportunities for abstractions in WMS for using efficient data management techniques. The traditional POSIX file I/O model can be replaced by scientific data formats. Additionally, data reduction and online analysis through streaming can be used in combination with policy-based usage of the tiered storage that includes SSDs, parallel file systems (PFS), and data stores (DS).

Using Scientific Data Management Libraries. To prevent a large ensemble from creating millions of files, applications can use high-performance, scientific data management libraries such as ADIOS [18] or HDF5 [19] that provide a custom self-describing file format for storing data. Data stores [20] and user-space file systems such as DeltaFS [21] can also be used as an alternative to the file-based model of data storage. Most scientific libraries work on local workstations as well as HPC machines, which can provide a portable solution for managing large data. To adapt file-based ensembles to use more efficient data storage models, WMS must provide rich abstractions that allow for easy translation from traditional file-based models to scientific data formats.

Automatic Provisioning of the Storage Hierarchy. Modern supercomputers consist of tiered storage that is comprised of device memory, local or remote flash memory, disk-based parallel file systems, and tape storage. Machines such as Summit and the upcoming exascale machine at OLCF, Frontier, have a node-local NVM device that can be used as temporary storage for large data [22]. Other systems such as the Perlmutter system at NERSC provide burst buffer nodes that are shared amongst compute nodes [23]. WMS should transparently be able to use different storage layers for storing data, instead of using the shared file system directly. Temporary files can be stored on faster, flash-based storage so that they can be read back efficiently for post-processing.

Scalable Metadata Management. File formats such as JSON are widely used for metadata storage as it is a lightweight text-based format that is human readable. However, they are not intended for large-sized metadata. WMS must use more sophisticated mechanisms such as scalable databases or scientific data formats for storing metadata.

Online Data Analysis. Along with managing large data, expensive post-processing has long been a challenge for HPC applications. There is increasing focus on online and in situ data processing methods for science in which data is processed in memory without storing it on the file system first. Data is staged to remote resources for processing. Additionally, data reduction techniques help reduce the overall size of data [24]. This prevents expensive operations of storing and reading large data back, which can often be a bottleneck for science applications. WMS can help large ensembles by providing abstractions for online processing of data, along with options to apply data reduction methods to reduce the overall data size. For workflows that permit partial post-processing, a WMS can dynamically post-process a subset of data and remove temporary files to reduce the pressure on the file system.

VI. Related Work

There is a wide range of workflow management systems available for general science use cases as well for domain-specific ad hoc solutions [25]. Systems such as the RADICAL-Ensemble Toolkit (EnTK) [26], libEnsemble [27], Balsam [28] are systems that have been used for HPC-scale ensemble runs. These systems focus mainly on providing abstract composition interfaces and high task throughput execution of runs. They are commonly referred to as Pilot systems as they manage resources on behalf of the user and aim to efficiently use them.

The EnTK toolkit provides for scalable task execution by bypassing the job step commands such as 'jsrun' and 'srun'. Instead, they use the underlying process management interface such as PMIX [17] available on HPC machines. EnTK creates separate directories for different stages of an ensemble Run and manages metadata in several different files. The libEnsemble system executes job step commands to run tasks on compute resources. It integrates the Balsam [29] system such that libEnsemble worker processes run Balsam on compute nodes. This potentially provides better task scaling on large HPC systems. Systems such as HTCondor [15], Nextflow [30], Pegasus [31] provide the capability to dynamically schedule jobs, which can be useful for capacity-class ensemble designs. Other systems that provide high-throughput task execution are Parsl [32] which provides high throughput task executors on HPC and cloud platforms, the Flux system [33] which is a more general-purpose runtime system for scalable task execution, and Swift/T [34] which is a language for distributed parallel scripting. The ExaWorks project [35] aims to bring several of these WMS together to provide a common SDK to democratize workflow technologies. It provides a portability layer across different HPC workload managers to create portable workflows with a standard API.

In general, we observe that some of the challenges described in this paper are addressed partially by existing WMS, but tighter integration with policies around resource usage set by supercomputing facilities and managing large data remain a challenge going forward. An ecosystem of workflow tools that provides an abstraction that allows ensemble workflows to select desired features can help scale them on to large systems.

VII. Conclusion

In this paper, we describe challenges associated with scaling an ensemble workflow to extreme scales on a leadership class supercomputer. While the use of a Workflow Management System (WMS) helps utilize resources efficiently for ensemble runs, scaling the ensemble to several million runs presents challenges often overlooked by existing WMSs. The challenges faced include 1) memory and performance issues during campaign/ensemble composition, 2) limitations on the number of concurrent batch jobs in the ensemble, 3) limitations on the scalability of runs in a job, 4) file system and metadata scalability issues due to overhead imposed by the large number of files and directories, and 5) overhead of post-processing large volumes of data. To overcome these limitations, we discuss techniques to design the next generation of workflow tools and the abstractions necessary to compose and execute large ensembles in a scalable and portable way, ranging from workstations to supercomputers.

Acknowledgment

This research was supported in part by the Exascale Computing Project (17-SC-20-SC) of the U.S. Department of Energy (DOE), Scientific Discovery through Advanced Computing (SciDAC) program, and by DOE's Advanced Scientific Research Office (ASCR) under contract DE-AC02-06CH11357.

References

[1] Y. Okamoto, "Generalized-ensemble algorithms: enhanced sampling techniques for monte carlo and molecular dynamics simulations," *Journal of Molecular Graphics and Modelling*, vol. 22, no. 5, 2004.

[2] R. Chelli and G. F. Signorini, "Serial generalized ensemble simulations of biomolecules with self-consistent determination of weights," *Journal of Chemical Theory and Computation*, vol. 8, no. 3, 2012.

[3] S. Basu, K. Kumbier, J. B. Brown, and B. Yu, "Iterative random forests to discover predictive and stable high-order interactions," *Proceedings of the National Academy of Sciences*, vol. 115, no. 8, pp. 1943–1948, 2018.

[4] A. Cliff, J. Romero, D. Kainer, A. Walker, A. Furches, and D. Jacobson, "A high-performance computing implementation of iterative random forest for the creation of predictive expression networks," *Genes*, vol. 10, no. 12, p. 996, 2019.

[5] K. Mehta, B. Allen, M. Wolf, J. Logan, E. Suchyta, J. Choi, K. Takahashi, I. Yakushin, T. Munson, I. Foster *et al.*, "A codesign framework for online data analysis and reduction," in *2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*. IEEE, 2019, pp. 11–20.

[6] K. Mehta, B. Allen, M. Wolf, J. Logan, E. Suchyta, S. Singhal, J. Y. Choi, K. Takahashi, K. Huck, I. Yakushin, A. Sussman, T. Munson, I. Foster, and S. Klasky, "A co-design framework for online data analysis and reduction," *Concurrency and Computation: Practice and Experience*, 8 2021. [Online]. Available: https://www.osti.gov/biblio/1817542

[7] I. Yakushin, K. Mehta, J. Chen, M. Wolf, I. Foster, S. Klasky, and T. Munson, "Feature-preserving lossy compression for in situ data analysis," in *49th International Conference on Parallel Processing-ICPP: Workshops*, 2020, pp. 1–9.

[8] E. Suchyta, S. Klasky, N. Podhorszki, M. Wolf, A. Adesoji, C. Chang, J. Choi, P. E. Davis, J. Dominski, S. Ethier *et al.*, "The exascale framework for high fidelity coupled simulations (effis): Enabling whole device modeling in fusion science," *The International Journal of High Performance Computing Applications*, vol. 36, no. 1, pp. 106–128, 2022.

[9] S. Singhal, A. Sussman, M. Wolf, K. Mehta, and J. Y. Choi, "Dyflow: A flexible framework for orchestrating scientific workflows on supercomputers," in *50th International Conference on Parallel Processing Workshop*, 2021, pp. 1–11.

[10] "LSF Job Step Manager," https://www.ibm.com/docs/en/spectrum-lsf/10.1.0?topic=SSWRJV_10.1.0/jsm/jsm_kickoff.html, 2020.

[11] M. A. Jette, A. B. Yoo, and M. Grondona, "Slurm: Simple linux utility for resource management," in *In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003*. Springer-Verlag, 2002, pp. 44–60.

[12] United States Census Bureau, *American Community Survey: Data Profiles*, 2019, Accessed: Mar. 11 2021. [Online]. Available: https://www.census.gov/acs/www/data/data-tables-and-tools/data-profiles/2019/

[13] K. Walker and M. Herman, *tidycensus: Load US Census Boundary and Attribute Data as 'tidyverse' and 'sf'-Ready Data Frames*, 2021, r package version 0.11.4. [Online]. Available: https://CRAN.R-project.org/package=tidycensus

[14] "NERSC QOS Limits and Charges," https://docs.nersc.gov/jobs/policy/#qos-limits-and-charges.

[15] P. Couvares, T. Kosar, A. Roy, J. Weber, and K. Wenger, "Workflow management in condor," in *Workflows for e-Science*. Springer, 2007, pp. 357–375.

[16] W. Gropp, W. D. Gropp, E. Lusk, A. Skjellum, and A. D. F. E. E. Lusk, *Using MPI: portable parallel programming with the message-passing interface*. MIT press, 1999, vol. 1.

[17] R. H. Castain, J. Hursey, A. Bouteiller, and D. Solt, "Pmix: process management for exascale environments," *Parallel Computing*, vol. 79, pp. 9–29, 2018.

[18] W. F. Godoy, N. Podhorszki, R. Wang, C. Atkins, G. Eisenhauer, J. Gu, P. Davis, J. Choi, K. Germaschewski, K. Huck *et al.*, "Adios 2: The adaptable input output system. a framework for high-performance data management," *SoftwareX*, vol. 12, p. 100561, 2020.

[19] M. Folk, G. Heber, Q. Koziol, E. Pourmal, and D. Robinson, "An overview of the hdf5 technology suite and its applications," in *Proceedings of the EDBT/ICDT 2011 Workshop on Array Databases*, 2011, pp. 36–47.

[20] M. S. Breitenfeld, N. Fortner, J. Henderson, J. Soumagne, M. Chaarawi, J. Lombardi, and Q. Koziol, "Daos for extreme-scale systems in scientific applications," *arXiv preprint arXiv:1712.00423*, 2017.

[21] Q. Zheng, K. Ren, G. Gibson, B. W. Settlemyer, and G. Grider, "Deltafs: Exascale file systems scale better without dedicated servers," in *Proceedings of the 10th Parallel Data Storage Workshop*, 2015, pp. 1–6.

[22] "The Summit Supercomputer at OLCF," https://docs.olcf.ornl.gov/systems/summit_user_guide.html, 2020.

[23] "Using Perlmutter - NERSC Documentation," https://docs.nersc.gov/systems/perlmutter/.

[24] I. Foster, M. Ainsworth, J. Bessac, F. Cappello, J. Choi, S. Di, Z. Di, A. M. Gok, H. Guo, K. A. Huck *et al.*, "Online data analysis and reduction: An important co-design motif for extreme-scale computers," *The International Journal of High Performance Computing Applications*, vol. 35, no. 6, pp. 617–635, 2021.

[25] "Existing Workflow Systems," https://s.apache.org/existing-workflow-systems, 2020.

[26] V. Balasubramanian, M. Turilli, W. Hu, M. Lefebvre, W. Lei, R. Modrak, G. Cervone, J. Tromp, and S. Jha, "Harnessing the power of many: Extensible toolkit for scalable ensemble applications," in *2018 IEEE international parallel and distributed processing symposium (IPDPS)*. IEEE, 2018, pp. 536–545.

[27] S. Hudson, J. Larson, J.-L. Navarro, and S. M. Wild, "libensemble: A library to coordinate the concurrent evaluation of dynamic ensembles of calculations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 977–988, 2021.

[28] M. A. Salim, T. D. Uram, J. T. Childers, P. Balaprakash, V. Vishwanath, and M. E. Papka, "Balsam: Automated scheduling and execution of dynamic, data-intensive hpc workflows," *arXiv preprint arXiv:1909.08704*, 2019.

[29] M. Salim, T. Uram, J. Childers, P. Balaprakash, V. Vishwanath, and M. Papka, "Balsam: Automated scheduling and execution of dynamic, data-intensive HPC work flows," in *In Proceedings of the 8th Workshop on Python for High-Performance and Scientific Computing*, 2018.

[30] P. Di Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, and C. Notredame, "Nextflow enables reproducible computational workflows," *Nature biotechnology*, vol. 35, no. 4, pp. 316–319, 2017.

[31] E. Deelman, K. Vahi, G. Juve, M. Rynge, S. Callaghan, P. J. Maechling, R. Mayani, W. Chen, R. F. Da Silva, M. Livny *et al.*, "Pegasus, a workflow management system for science automation," *Future Generation Computer Systems*, vol. 46, pp. 17–35, 2015.

[32] Y. Babuji, A. Woodard, Z. Li, D. Katz, B. Clifford, R. Kumar, L. Lacinski, R. Chard, J. Wozniak, I. Foster, M. Wilde, , and K. Chard, "Parsl: Pervasive parallel programming in Python," in *HPDC'2019*, 2019.

[33] D. H. Ahn, J. Garlick, M. Grondona, D. Lipari, B. Springmeyer, and M. Schulz, "Flux: A next-generation resource management framework for large HPC centers," in *2014 43rd International Conference on Parallel Processing Workshops*, Sep. 2014, pp. 9–17.

[34] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster, "Swift: A language for distributed parallel scripting," *Parallel Computing*, vol. 37, no. 9, pp. 633–652, 2011.

[35] A. Al-Saadi, D. H. Ahn, Y. Babuji, K. Chard, J. Corbett, M. Hategan, S. Herbein, S. Jha, D. Laney, A. Merzky *et al.*, "Exaworks: Workflows for exascale," in *2021 IEEE Workshop on Workflows in Support of Large-Scale Science (WORKS)*. IEEE, 2021, pp. 50–57.