

ARTICLE TYPE

A Co-design Framework for Online Data Analysis and Reduction[†]

Kshitij Mehta*¹ | Bryce Allen² | Matthew Wolf¹ | Jeremy Logan¹ | Eric Suchyta¹ | Swati Singhal⁴ | Jong Youl Choi¹ | Keichi Takahashi⁵ | Kevin Huck³ | Igor Yakushin² | Alan Sussman⁴ | Todd Munson² | Ian Foster^{2,6} | Scott Klasky¹

¹Computer Science and Mathematics
Division, Oak Ridge National Laboratory,
Tennessee, USA

²Mathematics and Computer Science
Division, Argonne National Laboratory,
Illinois, USA

³Oregon Advanced Computing Institute for
Science and Society, University of Oregon,
Oregon, USA

⁴Department of Computer Science,
University of Maryland, College Park,
Maryland, USA

⁵Division of Information Science, Nara
Institute of Science and Technology, Nara,
Japan

⁶Department of Computer Science, The
University of Chicago, Illinois, USA

Correspondence

*Kshitij Mehta, One Bethel Valley Rd., Oak
Ridge, TN 37831-6057. Email:
mehtakv@ornl.gov

Present Address

One Bethel Valley Rd., Oak Ridge, TN
37831-6057

Abstract

Science applications preparing for the exascale era are increasingly exploring in situ computations comprising of simulation-analysis-reduction pipelines coupled in-memory. Efficient composition and execution of such complex pipelines for a target platform is a co-design process that evaluates the impact and tradeoffs of various application- and system-specific parameters. In this paper, we describe a toolset for automating performance studies of composed HPC applications that perform online data reduction and analysis. We describe Cheetah, a new framework for composing parametric studies on coupled applications, and Savanna, a runtime engine for orchestrating and executing campaigns of co-design experiments. This toolset facilitates understanding the impact of various factors such as process placement, synchronicity of algorithms, and storage vs. compute requirements for online analysis of large data. Ultimately, we aim to create a catalog of performance results that can help scientists understand tradeoffs when designing next-generation simulations that make use of online processing techniques. We illustrate the design of Cheetah and Savanna, and present application examples that use this framework to conduct co-design studies on small clusters as well as leadership class supercomputers.

KEYWORDS:

Exascale, Cheetah, Savanna, CODAR, workflows, in situ, online, reduction, co-design

1 | INTRODUCTION

High-performance computational science applications commonly operate as disk-mediated pipelines. A *simulation* runs in parallel and outputs state information to persistent storage. Separate *analysis* programs then read the data to extract scientifically useful information. This mode of operation is expected to become increasingly untenable at exascale, planned for 2020–2025. Exascale systems will compute at 10^{18} operations/sec, $100\times$ faster than today, but output speeds are likely to remain around 10^{12} bytes/sec¹—reducing the amount of data that can be saved to persistent storage per unit computation by a factor of 10–100. This dramatic reduction in relative I/O capability demands a change in the scientific computing process to produce a new class of coupled *simulation-analysis-reduction* computations that perform all three activities *at the same time*, in order to identify and

[†]Notice: This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (<http://energy.gov/downloads/doe-public-access-plan>).

output only scientifically important data. In the absence of this change, many scientific teams will not obtain usable results from exascale computers.

The simulation-analysis-reduction model we described is a part of a broader Online Data Analysis and Reduction (ODAR) motif². This ODAR motif represents a connection point between stream computing approaches and traditional, monolithic HPC application design. As can be seen in Figure 1, the requirement to construct HPC applications using an online motif can arise from a variety of situations. A central issue with the rise of this motif, however, is that combining high performance requirements with multi-application, synchronous runtimes creates breaks in the assumptions that most HPC tools and runtimes (e.g., MPI runtimes, schedulers, performance tools) make about what constitutes ‘a run’. As such, there are needs to find new tools and approaches that can address the need to design flexible, high performance, ODAR science applications.

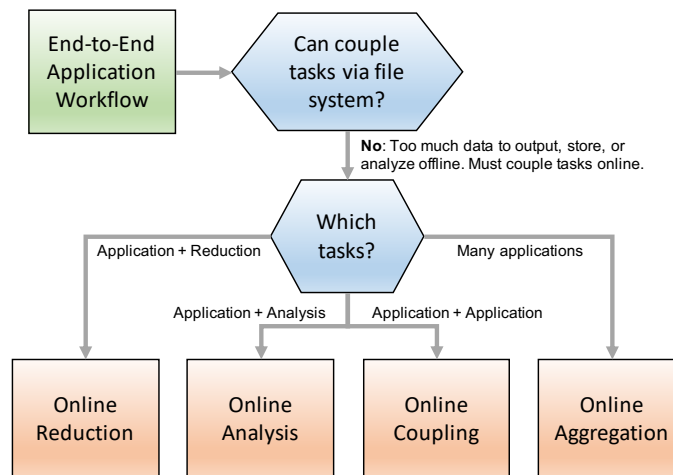


FIGURE 1 Motifs for Simulation-Analysis-Reduction workflows

The work we present here on the Cheetah and Savanna workflow environments demonstrates our initial steps towards developing the larger ecosystem of ODAR tools and run times. As a particular technical example, consider the growing I/O challenges for coupled and online science applications. While the multi-physics and multi-scale requirements of the science have driven the change to ODAR algorithms, more complex computer architectures (e.g., multicore nodes, heterogeneous nodes, NVRAM) and application structures (e.g., coupled simulations, uncertainty quantification, deep learning) make their efficient implementation yet more challenging. Coupled applications must be able to run multiple components simultaneously, with simulation component(s) producing data via computationally intensive processes, analysis component(s) identifying and processing events or characteristics of scientific interest within those data, and reduction component(s) organizing and compressing scientifically interesting information for output. As each component can be implemented, configured, connected, and mapped to computer systems in different ways, and those computer systems can potentially be adapted to better suit coupled computations, the number of design alternatives quickly becomes extremely large.

These considerations motivated the establishment of the Co-design Center for Online Data Analysis and Reduction (CODAR)³ in 2017, with the overall goal of enabling the principled co-design⁴ of computer systems, systems software, analysis methods, reduction methods, and applications to enable coupled simulations that can exploit the abundant computational power of exascale machines while adapting to their I/O limitations. Such a co-design involves exploring a large parameter space across application, middleware, and system layers. Managing a large campaign of experiments, which involves testing various complex orchestration and process placement options, along with various reduction methods on different architectures is highly challenging for end users, and it necessitates the use of specialized tools dedicated for running co-design campaigns on a variety of systems.

The primary goal of a co-design campaign is to aid end users in setting up their production runs in such a way as to obtain the best performance on the target architecture of the day. To achieve this goal, a catalog of performance information for sets of parameter combinations is desired. The catalog should reflect the choices faced by users in composing and orchestrating their workflows, and may encompass various performance metrics, including I/O bandwidth, computational throughput, and data fidelity. The results can provide the basis for creating a decision space to inform users of parametric combinations that meet their Quality of Service (QoS) requirements. Figure 2 provides an illustration of this co-design process.

The Cheetah and Savanna workflow composition and development tools are our contributions to addressing such needs for creating and running co-design campaigns for ODAR-motif applications. Our central model revolves around our idea of a co-design *campaign*, which is a workflow of workflows to explore a wide parametric space. In this context, a *Campaign* refers to a set of related efforts, often involving false starts, changes to codes, preliminary data analysis and visualization, etc., as well as the artifacts that accompany these efforts. Campaigns are extended periods of work on a focused set of topics, and are usually executed in the form of multiple batch jobs run over a period of days or weeks or months. Our elevation of the campaign as a core workflow context is central to our contributions, and the details are explored in Section 3. Cheetah is a campaign composition framework and experiment harness which provides an abstract specification format for creating a campaign of coupled Simulation-Analysis-Reduction workflows. Savanna is a runtime engine and in situ framework which orchestrates and executes complex workflows on the target platform.

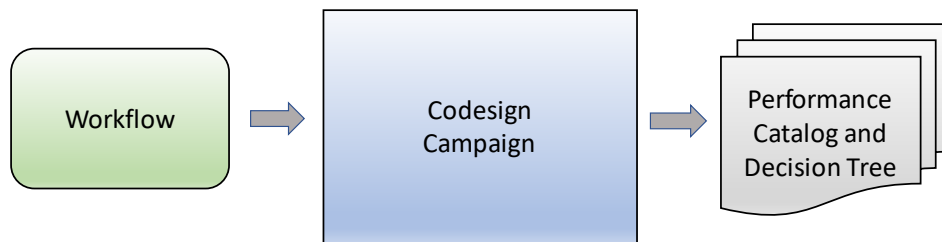


FIGURE 2 The output of a co-design study is a performance catalog / decision tree that lists different parameters and their associated costs. Users can utilize this information to setup production runs on a target machine depending on the Quality-of-Service (QoS) requirements.

In the following sections, we will further describe how Cheetah serves as a co-design campaign composition and management tool, as well as how the Savanna runtime dispatcher and controller works to enable not only co-design experimental campaigns but also a wider set of ODAR application runtime requirements. After detailing some of the implementation and design details in Section 3, we demonstrate the range of application scenarios and extensions to the core co-design application space that Cheetah and Savanna support in Section 4, before discussing related work and our conclusions.

2 | CHALLENGES OF CO-DESIGNING ONLINE WORKFLOWS

Generally, the Simulation-Analysis-Reduction (S-A-R) pipeline is a convenient way to represent a range of multi-application scientific workflows. For example, generating atomic-scale data and then analyzing those data to identify where particular crystals form in the atomic matrix corresponds to a particular simulation-to-analysis pairing. However, the general approach has been to treat these workflows as either post-hoc, interactive analyses, or as specialty, custom-coded environments that apply to a specific code base or even a particular scientist.

The challenge that Cheetah addresses is the continued evolution of this ecosystem away from custom-coded, bespoke solutions and towards shared tools and environments for Simulation-Analysis-Reduction pipelines. For it to be worthwhile to do *co-design campaigns*, there must be a broader base of usability and reusability of the pipeline. Motifs of execution patterns that can be applied multiple times in different simulation contexts can benefit from a decision tree of optimization choices (e.g., “which algorithm best sorts data on this kind of GPU?”). If it is just a performance tuning operation for a particular algorithm on a particular piece of hardware, then you do not need the full infrastructure of a co-design toolkit.

To this end, the CODAR project has targeted identifying key pipeline motifs that are relevant to the U.S. Department of Energy’s Exascale Computing Project’s portfolio. Figure 1 shows four such motifs: Online Reduction, Online Analysis, Online Coupling, and Online Aggregation. Science applications may exhibit properties of one or more motifs. For example, as shown in Figure 3 the Fusion Whole Device Modeling application (WDMApp) not only couples two different physics codes (Online Coupling), but also analyzes the joined output of that data (Online Analysis) and applies compression to reduce total output size (Online Reduction) ⁵. Complex workflows like these are expected to become more commonplace in the exascale and

post exascale era, and require exploring a large parameter space. Additionally, the vastly different architectures of prominent supercomputers, along with the advent of heterogeneous architectures, makes this a challenging task.

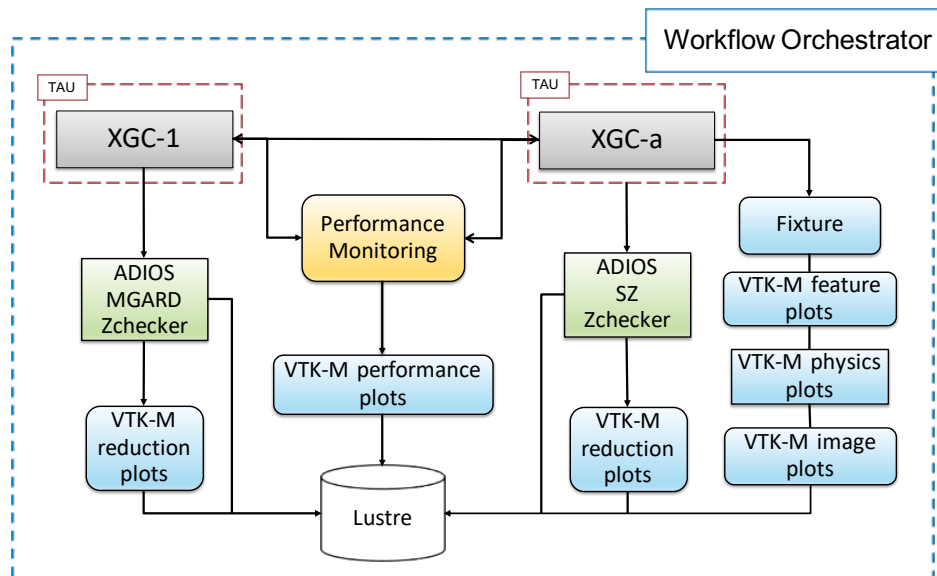


FIGURE 3 The Whole Device Modeling (WDM) workflow comprising the XGC-1 and XGC-a applications, coupled via the ADIOS middleware library.⁵ Data streams generated by each simulation are compressed with different methods and the compression error is analyzed by using the Z-checker library. The Fixture tool is used to detect features in the data stream. The simulations are instrumented with TAU and the performance data can be visualized online. VTK-m is used to visualize different output data streams.

The goal of a co-design campaign is to answer the following questions:

- Q1: What are the best data analysis and reduction algorithms for different classes of applications, in terms of speed, accuracy, and resource requirements? How can we implement those algorithms to achieve scalability and performance portability?
- Q2: What are the trade-offs in data analysis accuracy, resource needs, and overall application performance between using various data reduction methods to reduce file size prior to offline data reconstruction and analysis vs. performing more online data analysis? How do these trade-offs vary with exascale hardware and software choices?

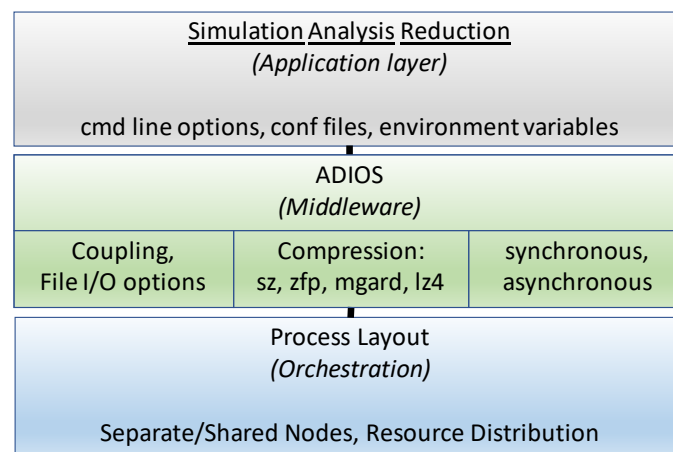


FIGURE 4 Three categories of parameters that can influence co-design campaigns. Application-level parameters directly control the application. Middleware parameters effect file I/O, staging, coupling options, and compression. Orchestration-level parameters control process placement and resource distribution between applications.

Figure 4 sketches some of the many degrees of freedom over which a co-design campaign would have to operate in order to answer these types of questions. The software engineering practices needed to support the creation of such workflows requires a whole set of tools for the connection and performance evaluation of the individual components, as well as the support for the design of experiments we offer through Cheetah. Here, we are able to take advantage of the rich software ecosystem around the Adaptable I/O System (ADIOS)⁶ and the Tuning and Analysis Utilities (TAU)⁷ packages. The ADIOS interface design allows for the easy transition from a post-hoc workflow construction into an online pipeline environment. ADIOS is based upon a publish-subscribe framework that provides file I/O primitives, along with staging transports for coupling applications in memory. Coupling can be performed using its Sustainable Staging Transport (SST), Strong Staging Coupler (SSC), and the DataMan wide-area staging engines. ADIOS also includes an interface to transform data produced by a simulation through various lossy and lossless compression methods, such as ZFP⁸, SZ⁹, MGARD^{10,11}, and LZ4¹².

Furthermore, even when a pipeline has been moved to online mode, there are considerable complexities in deciding how to place and orchestrate the individual MPI tasks across the available resources, as shown in Figure 5. TAU and ADIOS have built online monitoring and interoperability layers to further support the scientists' development needs for these sorts of deployments. As such, a key component of the Cheetah co-design framework is in presenting the available configurations for these packages as user-friendly parameters. This focus on higher-level models supports the engagement of computational and computer scientists in using a more rigorous co-design process to enable reusability and performance portability of application pipelines that analyze and reduce data online.

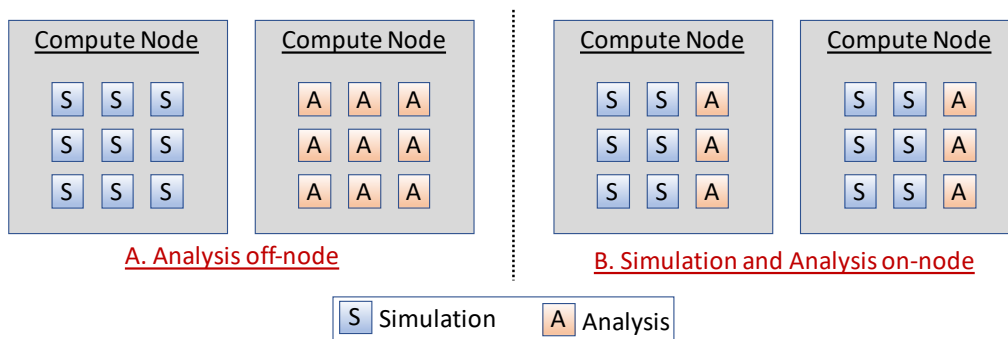


FIGURE 5 Process placement options for the simulation and analysis applications. Fig. A shows a scenario where analysis is performed off-node. Data is transferred across the network to a separate set of nodes dedicated for analysis. Fig. B shows the simulation and analysis applications sharing a node. Data is communicated through node-local memory. The distribution of compute resources for both scenarios has a significant impact on overall performance.

3 | A CO-DESIGN FRAMEWORK FOR ONLINE ANALYSIS

Cheetah and Savanna provide a framework for running large co-design parametric explorations through a rich abstraction based upon the campaign model for running a workflow of workflows. They hide low-level details from the user about the campaign creation, campaign execution, resource management, and scheduler options for complex process placement. That is, users specify *what* they want to test, rather than *how* it must be set up. An abstract end-to-end process of composing, executing, and inspecting a campaign is shown in Figure 6.

The core capabilities of a campaign-aware workflow framework are listed below.

1. *Campaign Composition* – A high-level specification format to compose campaigns and instantiate them
2. *Interoperability* – An abstract campaign manifest for interoperability between workflow runtimes
3. *Campaign Execution* – A runtime engine to run the campaign on a target machine
4. *Campaign Monitoring* – The ability to track the progress of a running campaign
5. *Performance Introspection* – The ability to generate a performance report for a completed campaign

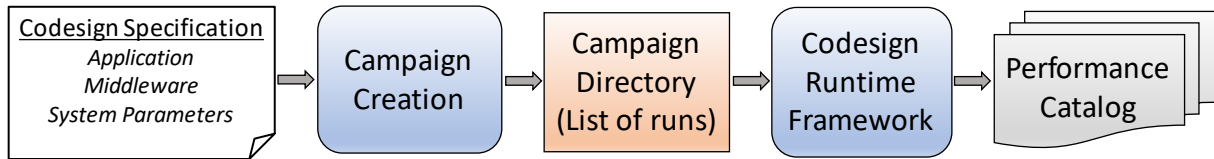


FIGURE 6 A functional overview of a toolset for conducting co-design studies. A specification written in a high-level language is used to generate a campaign on a target system. A runtime framework executes the campaign and generates a performance catalog of the campaign.

In order to demonstrate the importance of each of these capabilities for campaign management, the following sections 3.1 - 3.5 showcase the Cheetah-Savanna system's approach to each of them in turn. Cheetah—the composition layer of the framework—is an experiment harness for composing and executing campaigns of parameter sweeps for a set of coupled applications for studying different combinations of online data analysis orchestration schemes. It provides an abstract specification format for creating a campaign consisting of parameter sweeps and a directory schema with independent workspaces for the different parameter combinations. Savanna—the execution layer—translates an abstract campaign definition into experiments that can be run on the target system. It is the internal runtime engine that provides the low level implementation of the specification. The relationship between them is demonstrated in Figure 7.

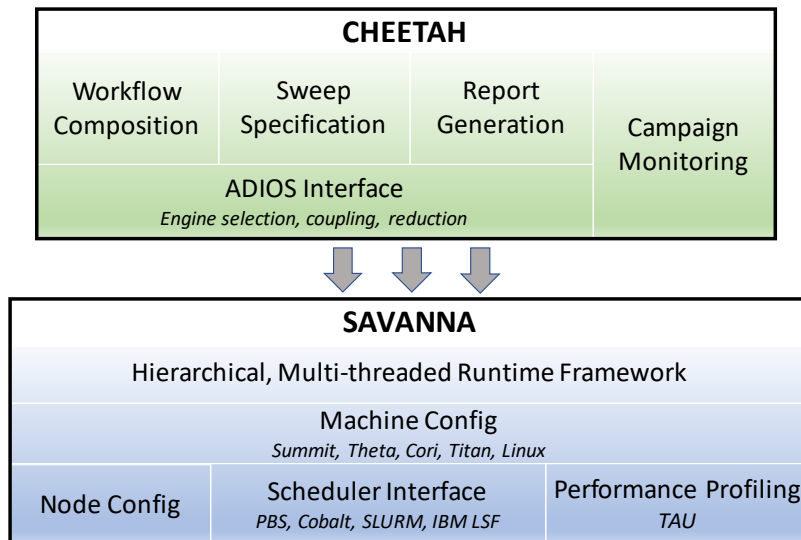


FIGURE 7 The architecture of the Cheetah-Savanna toolset as a means for a composition and execution framework for co-design campaigns. Cheetah provides the high-level specification to create a campaign. Savanna provides the runtime framework to execute the campaign on the target system.

3.1 | Campaign Composition

The Cheetah object model is centered around the concept of a *Campaign*. As shown in Figure 8, a campaign is a workflow of workflows or, alternatively, a collection of experiments/runs. Cheetah's campaign model provides multi-user support – runs from multiple users can be part of the same campaign.

The campaign abstraction has proven to be useful for organizing artifacts, such as application codes and data, around the typical working patterns of high-performance computing¹³. The data products of a campaign are typically filtered and cherry-picked to produce publication-quality datasets, but generally not released in their entirety. By providing explicit support for campaigns, Cheetah allows a set of individual experiments to be grouped and their results easily combined. A campaign-based approach allows conducting a methodical search for a parameter space that satisfies core constraints. Campaign-based studies

include searching for parameter combinations that reduce cost of data movement in a large scale application, or searching for configurations that lead to optimal resource utilization, or searching for a parameter space that satisfies constraints in a machine-learning application pipeline, to name a few. Cheetah's fundamental object model provides the core infrastructure for conducting such studies through its Campaign design. In addition to support for the campaign workflow construction, there is also a need to construct the details of the encompassing experimental workflows, based on standard existing patterns as well as novel ones better suited to S-A-R scenarios.

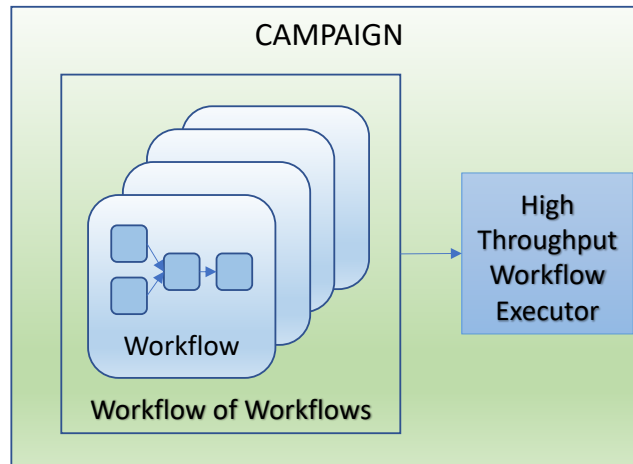


FIGURE 8 The Campaign model for co-design studies. A campaign is a workflow of workflows setup through a parametric description. Each workflow can be a collection of application components, in which some components run concurrently, while some run as a pipeline. A high-throughput workflow executor runs the campaign on a target system.

New Workflow Pattern: The Bundled Workflow

Simulation-Analysis-Reduction workflows involve running simulation and analysis codes concurrently. In MPI terminology, they are part of distinct `MPI_COMM_WORLD` communicators. They may progress independently of each other and may exchange data via streaming. We introduce a new execution pattern called ‘bundled’ workflows in which two or more applications are launched as a single *bundle*. A bundle is a collection of applications that are launched independently by a workflow manager, but must run concurrently in order to exchange data. A bundle cannot be launched partially; all applications that form the bundle must be launched concurrently. Thus, care must be taken to ensure sufficient resources are available to run the entire bundle.

In addition to bundled workflows, Cheetah also supports the application pipeline motif as a first-class citizen. While bundles are a collection of concurrently running application components, a pipeline is a set of applications with dependencies between them. Based upon the target system, Cheetah calculates the resources required to run bundles and pipelines, and thus manages launching bundles and pipelines on the provisioned resources.

Real science codes may combine these workflow motifs in various ways to achieve their end goal. For example, a subset of application components in a workflow may run concurrently (e.g., XGC-1 and XGC-a in Figure 3), whereas some components may depend on the outcome of others (various VTK-M plots in Figure 3). The campaign abstraction for workflow specification needs to be robust enough to handle both the pure motif examples and the hybrid ones equally.

3.1.1 | Campaign Specification

Cheetah provides a high-level specification format in the Python programming language for composing a campaign. An abstract specification presents a unified way to construct campaigns on various machines, as opposed to writing ad hoc scripts and managing experiments manually. The campaign specification is used to describe parametric evaluations for online data analysis of coupled applications, along with the ability to group experiments into collections. A campaign is particularly void of low-level

system and scheduler details. As different job schedulers such as Slurm¹⁴, Cobalt¹⁵, PBS, and IBM LSF are used on different supercomputing facilities, a common abstraction to build experiments proves highly useful.

The specification format consists of three categories of configuration options: applications settings, global campaign settings, and the parameter sweeps of the campaign.

Application Settings

Application settings provide a way for users to describe application components of a workflow. It defines how applications are run, location of their executables, input files required, and their ADIOS configuration file. The section on Cheetah's object model describes how Cheetah provides a *Param* interface to configure and run applications. For example, a *ParamCmdLineArg* can be used to represent a command line argument to run a campaign. Users are not required to instrument or modify their application source codes to use Cheetah. As a workflow tool, Cheetah interacts with the application in an external manner without impacting the application's execution. By default, applications are launched as part of a top-level bundle. Applications with dependencies between them form a pipeline.

Campaign Settings

Campaign settings consist of global options such as global runtime environment setup, post-processing scripts, and user account settings. It includes options for failure handling to control an experiment if one of more underlying components in an experiment exhibits a failure. Additionally, users can specify pre- and post-processing functions that are applied to the campaign.

The Parameter Sweep Object Model

Cheetah's model for creating parameter sweeps is shown in Figure 9. A Campaign is a collection of SweepGroups, which are collections of Sweeps. A two-level hierarchy allows for better logical grouping and also provides the underlying infrastructure for dynamic control of a campaign. SweepGroups provide a logical way of grouping experiments that explore a common category of co-design experiments. For example, users may group parameters for tuning communication overhead in a SweepGroup, whereas parameters for exploring different file I/O techniques may be grouped in a separate SweepGroup. Users can compose application bundles or construct pipelines of applications using the SweepGroup's dependency options for creating a DAG (directed acyclic graph) of applications.

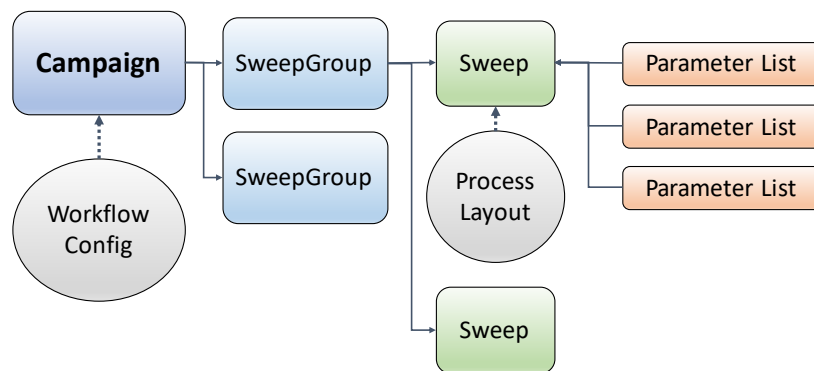


FIGURE 9 The Cheetah parameter sweep object model. A top-level Campaign object is a collection of SweepGroups, which are a collection of Sweeps. A Parameter Sweep is a collection of parameters and values that must be explored as part of a co-design study.

A SweepGroup represents a batch job on the underlying system. Properties such as resource size of the allocation, walltime limit, the timeout for each experiment etc. are set in the SweepGroup properties. Applications can also be launched using the MPMD (multiple program, multiple data) paradigm. Experiments in the SweepGroup can be marked for automatic performance profiling and tracing using the TAU toolkit¹⁶. Users may also setup multiple trials for each experiment to obtain multiple readings for each data point and conduct statistical analysis on the results.

A fundamental feature of Cheetah is the ability to *sweep* over parameter values that need to be explored in the study. A Sweep is a collection of parameters created as different *Param* object types and a list of values for each parameter that the user is interested in exploring. Param types represent different application-, middleware-, and orchestration-level parameters. For example, *ParamCmdLineArg* is used to represent a command line parameter, *ParamRunner* is used for setting the degree of parallelism, *ParamEnvVar* represents sweeping over environment variables, *ParamConfig* is used to set key-values pairs in configuration files, and so on. Cheetah also provides the *ParamADIOSXML* parameter for setting up and tuning various engines for coupling, data staging, and file I/O in ADIOS XML configuration files. This parameter can be used to set transport engines, engine parameters, and reduction operators to tune an application's ADIOS settings. As ADIOS provides a convenient way to configure various aspects of I/O, coupling, and data reduction through a settings file, it is relatively straightforward in Cheetah to sweep over ADIOS parameters through a Param type that can modify the settings file. Other I/O libraries can be supported in a similar way in Cheetah by creating new Param types. Listing 1 shows a example specification file with different Param types.

3.1.2 | The *VirtualNode* Abstraction for Fine-Grained Process Placement

A S-A-R workflow such as the one shown in Figure 3 can easily suffer from performance issues arising due to communication bottlenecks and other inefficient utilization of resources. For managing communication overhead, the topology of process placement can have a strong impact. Additionally, different application components have different resource requirements. Simulations may make heavy use of GPUs, whereas some analysis codes may only utilize the CPUs on a node. One of the parameters in a co-design exploration is exploring different techniques for resource utilization. This includes topological exploration, which evaluates the performance of various strategies for mapping processes to resources.

It is common practice to spawn concurrently running applications of a workflow on separate compute nodes of a cluster. However, this often leads to sub-par performance and resource utilization. This is especially applicable for machines such as the Summit supercomputer at Oak Ridge National Laboratory, in which a single compute node consists of 6 GPUs and 42 CPU cores. The main simulation may be tuned to utilize the GPUs on a node. However, CPU cores are often idle, and analysis applications spawned on separate nodes can largely under-utilize node resources, thereby leading to under-utilized compute resources and high communication overhead.

Effective orchestration of a coupled workflow requires more sophisticated process mapping techniques. System schedulers on HPC machines provide limited support for complex process placement. Users can control the degree of parallelism, however, sharing compute nodes between ranks from different MPI applications is cumbersome. In §4, we show examples in which co-locating simulation and analysis ranks on a compute node such that the analysis ranks are mapped to idle cores on a compute node shows significant performance advantages.

To develop an interface for process placement that is transparent of the underlying scheduler interface, the following options were considered.

1. A simple interface with options to specify the total number of ranks and ranks per node for each application (see *ParamRunner* interface in Listing 1)
2. An interface that employs Resource Sets as a common mechanism across all systems
3. An interface where users can manually map MPI ranks to resources on a compute node

We explore the pros and cons of these approaches below.

The *ParamRunner* interface

As noted previously, a *ParamRunner* parameter is used to setup the number of processes for an application. This can be extended to include finer options such as number of ranks per node etc. However, they do not suffice for use on heterogeneous architectures as it requires more explicit mapping between ranks and resources such as GPUs.

The *Resource Set* interface

The IBM LSF job scheduler and workload management solution on Summit introduces the notion of 'Resource Sets' which is a mapping of MPI ranks to resources on a compute node. A resource set allows users to configure a set of compute and memory resources within a compute node. Users then specify the number of resource sets per node and the total number of resource sets

```

class ReactionDiffusion(Campaign):
    name = "ReactionDiffusion"

    # DEFINE APPLICATIONS
    codes = [
        ("sim", dict(exe="prod.py", adios_xml_file='adios2.xml')),
        ("mean_calc", dict(exe="m_calc.py", adios_xml_file='adios2.xml'))
    ]

    # CAMPAIGN SETTINGS
    supported_machines = ['local', 'theta', 'summit']
    run_post_process_script = None
    scheduler_options = {'summit': {'project':''}}
    app_config_scripts = {'summit': 'env_setup.sh'}

    # PARAMETER SWEEPS
    sweep1_parameters = [ # sweep over list values
        ParamRunner      ('sim', 'nprocs', [2048,4096]),
        ParamRunner      ('mean_calc', 'nprocs', [128]),
        ParamCmdLineArg  ('sim', 'size_per_pe', 1, ['1M','2M','4M']),
        ParamKeyValue    ('sim', 'steps','settings.conf', 'steps', [10,25,50,100]),
        ParamADIOS2XML   ('sim', 'producer', 'engine', [ {"SST": {}} ]), # Coupling
        ParamEnvVar      ('sim', 'openmp', 'OMP_NUM_THREADS', [1,4]),
    ]

    # Create a sweep
    sweep1 = Sweep ( parameters = sweep1_parameters, rc_dependency=None )

    # Create a sweep group from the above sweep(s).
    # Each group is submitted as a separate job.
    sweepGroup1 = SweepGroup (
        "sg-1",
        walltime=300,
        per_run_timeout=60,
        parameter_groups=[sweep1],
        launch_mode='default', # Or MPMD
        # optional:
        nodes=10, # Calculated automatically if not specified
        tau_profiling=True,
        component_inputs = {'sim': ['some_input_files'], }
    )

```

Listing 1: Layout of a Campaign Specification File

to run an experiment. This is a significant deviation from the older scheduler interface of specifying the total number of MPI processes and the number of MPI processes per node.

To develop a consistent interface across supercomputers, we initially considered an interface that uses Resource Sets as the mechanism to setup a workflow. A Resource Set would describe the minimal, atomic set of resources on a compute node required for an application. On homogeneous architectures consisting of CPUs only, a Resource Set would default to a single core. For complex heterogeneous architectures, it would be a mapping of application ranks to CPUs and GPUs using various command

line arguments. However, a major disadvantage of this approach is that configuring Resource Sets to share a compute node between multiple applications using different arguments for binding and distributing ranks easily becomes confusing and error-prone. Consider a coupled workflow similar to the one shown in Figure 3 which consists of many applications. This workflow may be setup in many different ways where ranks from all or a subset of all applications may share compute nodes. Setting up this workflow using the Resource Set interface for each application is tedious, non-intuitive, and highly susceptible to errors.

The *Virtual Node* interface

Given the above complexities, we chose instead to develop an interface where users can bind processes to a compute resource manually. We have developed a high-level abstraction that presents a ‘view’ of a compute node of the target machine with a list of compute resources (CPUs and GPUs) on the node. This novel interface, called a *Virtual Node*, provides a programmatic way to describe the layout of application processes on compute nodes. Users create a *Virtual Node* object for the target system and explicitly map application ranks to resources on the node. All experiments in a *Sweep* inherit the process layout described in the *Virtual Node*. Based on the total number of processes requested for each application, Cheetah automatically calculates the total number of nodes that will be required to run the experiment.

Listing 2 shows an example of a *VirtualNode* for the Summit supercomputer. It describes a node layout in which simulation and analysis ranks are co-located on a compute node. It maps simulation ranks such that two groups of 10 processes each are placed on consecutive cores with a stride of 12 cores between them. An analysis process is placed at the end of each group of simulation processes. This custom virtual node is replicated depending on the total number of ranks spawned for the simulation and analysis codes. The *VirtualNode* interface provides great flexibility and clarity for obtaining fine-grained placement of processes from multiple applications on to compute resources. §4.1 shows different ways of using the *VirtualNode* interface to obtain process mappings, such as placing different application processes on different nodes, co-locating compute nodes, and co-locating CPU sockets on compute nodes. It forms one of the most powerful features of Cheetah, as the capability to obtain such complex process placement can be instrumental for performance optimization on upcoming exascale machines.

3.1.3 | Instantiating a Campaign

Once a specification is created, users invoke Cheetah to create a campaign endpoint on the target system. A campaign is a self-describing collection of independent experiments with their own, isolated workspaces. Each *SweepGroup* in the Campaign is a batch job that can be launched independently of other *SweepGroups*. Parameter sweeps in a *SweepGroup* are serialized into a list of experiments. The campaign schema permits multiple users to create and execute experiments within a single campaign endpoint. Users are provided with their individual workspaces. This model allows multiple users to conduct a co-design campaign under a single campaign endpoint, thereby enabling a holistic view of the entire multi-user campaign.

Similarly, experiments in a campaign are independent of other experiments and are provided their own workspace. This allows multiple experiments to run concurrently and safely depending on the available resources. As will be seen in the following sections, Cheetah and Savanna automatically calculate the resource requirements for each application and experiment in the campaign, and can thus manage sophisticated execution of experiments.

3.2 | Interoperability

Although interoperability can have an array of colloquial meanings, for our work we have chosen to narrow the focus specifically to achieve an exchange between workflow runtimes. We have investigated constructing an abstraction of the campaign manifest that will be sharable and transportable. In our reference implementation, Cheetah converts the Python-based specification into a JSON-based manifest of experiments upon instantiation. This custom metadata format, termed the Functional Object Bundle (FOB) format, is a self-describing medium for describing a campaign. It fully describes the campaign, which includes information about the enclosed applications, parameter sweeps and *SweepGroups*, *VirtualNode* setups, pre- and post-processing tasks, and global campaign settings. The relationship of the FOB to these campaign components can be seen in Figure 10.

The FOB provides an interoperable representation to connect the composition and execution layers of the framework. Our execution engine, Savanna, provides a simple user-level job runner that can parse our implementation of the FOB. Savanna’s support for the Simulation-Analysis-Reduction pipeline model is distinct from that used in other user-space resource managers such as Flux¹⁷, Swift/T¹⁸, Parsl¹⁹, and Radical-Pilot²⁰, although we see good internal interfaces for Savanna to work with these systems when required. The FOB was designed so that in our future work we could explore connecting high-level workflow

```

class SummitNode:
    def __init__(self):
        self.cpu = [None] * 42
        self.gpu = [None] * 6

shared_node_layout = SummitNode()

# Create 10 simulation ranks on each socket
for i in range(10):
    shared_node_layout.cpu[i] = 'sim:' + str(i)
for i in range(10):
    shared_node_layout.cpu[22+i] = 'sim:'+ str(10+i)

# Map gpus 0 and 3 to two sim ranks
shared_node_layout.gpu[0] = ['sim:0', 'sim:1']
shared_node_layout.gpu[3] = ['sim:10', 'sim:11']

# Create 2 analysis ranks on socket 2
shared_node_layout.cpu[38] = 'al:1'
shared_node_layout.cpu[39] = 'al:2'

# Provide the total number of ranks. The number of nodes required will be
# calculated automatically by Cheetah,
# and the Virtual Node will be replicated for each node
ParamRunner ('sim', 'nprocs', [5000, 10000])
ParamRunner ('al', 'nprocs', [128, 256])

```

Listing 2: A Virtual Node mapping showing simulation and analysis tasks mapped to node resources on Summit.

designs to other workflow engines, such as these, that might have different performance or features for other motifs (e.g., bag-of-tasks).

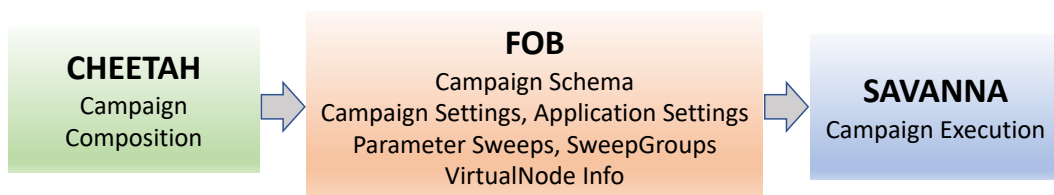


FIGURE 10 The JSON-based Functional Object Bundle (FOB) format as an interoperable representation of a campaign.

3.3 | Campaign Execution

Savanna is the runtime engine implementation which provides the execution layer of the framework. Savanna manages resources such that new workflows are submitted and tracked as currently running workflows complete and resources become available. It provides the constructs for running application pipelines on the target machine and a translation layer to convert the workflow specification to underlying job scheduler calls.

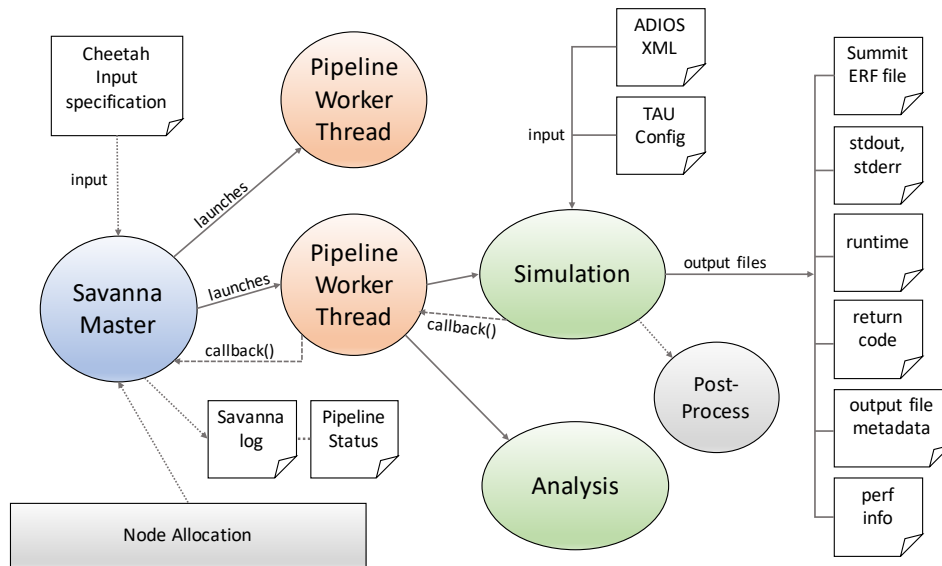


FIGURE 11 The Savanna runtime framework consists of a hierarchical, master-worker framework for launching experiments in a campaign. A master thread manages concurrent execution of co-design experiments by launching a worker thread for each experiment. The worker threads launch the application and manage performance information and experiment metadata.

Machine Configurations

Savanna provides the capability to express the architecture of a system in a `Machines` interface, which describes different characteristics of the system, such as the architecture of a compute node, the memory hierarchy and the job scheduler. New machine definitions can be created as new systems become available, along with an option for adding smaller, local clusters. For example, several DOE supercomputers such as Cori, Theta, and Summit are included in the default Savanna release, along with support for generic Linux workstations. Machine definitions allow Savanna to calculate the resource requirements of workflows. Transparently calculating resource requirements of a workflow of application bundles and pipelines is a strong feature of the Cheetah-Savanna toolset.

Architecture

Savanna implements a hierarchical, multi-threaded, master-worker framework to run a campaign of experiments, as shown in Figure 11. For each `SweepGroup` (batch job), a top-level Savanna instance is created that manages the group. This master thread reads the group's global manifest of experiments and starts running experiments according to the available resources until no more experiments can be run. It launches a Savanna worker thread for each experiment, and the worker in turn launches a thread for each component application in the workflow (simulation code, analysis code). An application's worker thread generates the scheduler commands dynamically to launch the application. The master manages the resource allocation for the job and assigns nodes to the co-design experiments. For example, if a user allocates 1000 nodes to a Sweep Group in which the experiments require a hundred nodes, Savanna executes ten experiments concurrently and manages the nodes allocated to each experiments. Recall that as each experiment runs in its isolated workspace and independent of other experiments, multiple experiments can be run concurrently in safe way. The worker threads communicate with the master to exchange information about the status of an experiment. Worker threads manage their experiment's status and performance profiles in the experiment's workspace. Additionally, they automatically capture the runtime for each component and create a metadata file for later inspection. All Savanna threads run on the login or service node depending on the underlying machine, and do not consume any resources on compute nodes. As they are lightweight threads that launch and monitor applications, their impact on the campaign runtime is considered minimal. When an experiment completes, Savanna selects the next available experiment from the campaign manifest and schedules it on the available compute nodes.

```
app 0: simulation_lammps
app 1: rdf_analysis

rank: 0: {host: 1; cpu:{0-3}; gpu: {0}} : app 0
rank: 1: {host: 1; cpu:{4-7}; gpu: {0}} : app 0
rank: 2: {host: 2; cpu:{8-11}} : app 0
rank: 3: {host: 2; cpu:{12-15}} : app 0
...
...
rank: 20: {host: 2; cpu:{0-3}; gpu:{0}} : app 1
rank: 21: {host: 2; cpu:{4-7}; gpu:{0}} : app 1
```

Listing 3: An ERF file on the Summit supercomputer that shows how MPI ranks are explicitly mapped to nodes, and to resources within a node.

Implementing the *Virtual Node Interface*

As stated before, Savanna translates the campaign specification and resource specification into actual job scheduler calls on a target system. Systems that do not support fine-grained resource mapping as expressed in the Virtual Node interface provide a basic set of features that allow setting the total number of MPI ranks and the number of ranks per node. These options are usually provided to the job scheduler as command line arguments. Savanna translates the campaign specification to set the job scheduler options to set these options as needed.

Systems that support fine-grained resource mapping require using custom features provided by the underlying job scheduler. On Summit, Savanna translates a Virtual Node into Explicit Resource Files (ERF)^{21,22} that are input to the *jsrun* utility for launching an application. The ERF file format is an exclusive feature developed by IBM that provides a mapping between CPUs, GPUs, memory, rank IDs, and hostnames for a running application. Listing 3 shows parts of an ERF file that map ranks of a simulation and analysis application to compute resources on different nodes. A Savanna worker thread dynamically generates an ERF file for an application based upon the nodes assigned to the experiment. On other architectures such as the Cori supercomputer at NERSC, the Virtual Node can be translated into a CPU map that is provided as a command line option along with environment variables to the SLURM job scheduling utility.

Savanna also provides support for different services such as profiling and tracing a workflow through the TAU performance toolkit. If profiling and/or tracing are enabled in a SweepGroup, Savanna sets the profiling environment by setting environment variables and creating output directories for each application in the workflow for storing its TAU profiles and traces. It then transparently launches the workflow using the TAU utility. The performance reporting engine described in Section 3.5 then aggregates profiling and tracing information to generate a summary profile for the user for inspection. For future versions of Cheetah, we plan on developing a more generic interface that allows other third-party performance profiling libraries and tools to be included in a campaign for performance analysis.

3.4 | Campaign Monitoring

Users have the ability to monitor a campaign through command lines tools included in the software package. The status of a running campaign can be queried to know that status of all sweep groups and individual experiments within them. The monitoring tool reports if an experiment is running, completed, or not started, along with a status label to denote successful completion, failure, or timeout. Additionally, users can resubmit a campaign to resume partially completed SweepGroups.

3.5 | Performance Introspection

Cheetah comes with a performance reporting tool that can be used to parse a completed campaign and extract performance information about all runs. The tool iterates over all the SweepGroups and the enclosed experiments, and collects various metadata and performance metrics. This performance catalog is presented to the end user as a CSV file.

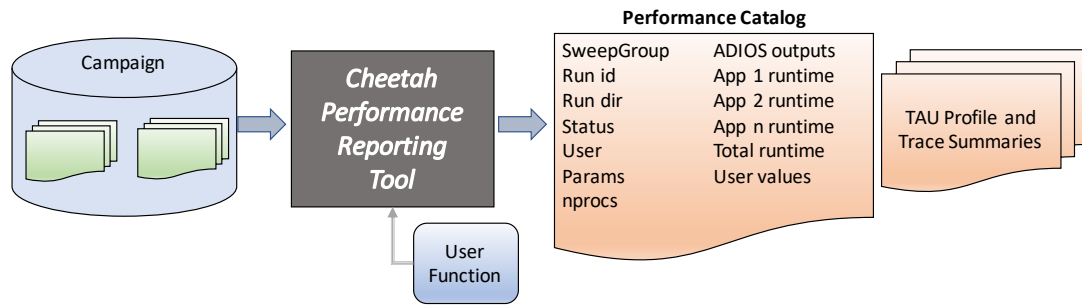


FIGURE 12 Generating a performance catalog from a completed Campaign. The Performance Reporting Tool in Cheetah traverses the entire campaign to generate a performance catalog containing metadata, runtimes of all experiments and their individual application components, ADIOS output files and their sizes, and user-generated information.

The performance catalog provides various performance metrics about experiments. This includes the runtime of a workflow and its component applications, sizes of output ADIOS data, run status (success/failure/timeout), and additional data extracted from the TAU profiling tool. Users also have the option of providing an external function to the performance reporting tool for extracting additional information from each experiment in the campaign. External functions are helpful for extracting information that may be contained in non-standard locations such as text-based application output files and error logs.

4 | APPLICATION EXAMPLES

As we have seen, the core motivation of having a co-design experimental harness for online and coupled computation examples led to a number of innovative contributions in the Cheetah and Savanna designs. The broad strokes of the campaign management paradigm that Cheetah employs are easy enough to understand in the context of a general description of the design, but some of the interesting complexities and capabilities are best seen through examples. In what follows, we show how the native support for Simulation-Analysis-Reduction (S-A-R) pipelines drawn from the ODAR Motif along with the campaign-oriented design enable a wide variety of current and future application scenarios.

The initial examples showcase Cheetah as a co-design harness at work through explorations of placement, use of non-volatile storage, and performance impacts. The compression and dynamic management studies both use extensions for autonomic controls within the Savanna runtime to understand the (positive) co-design implications of including dynamism within the S-A-R motif. Finally, we present two examples where the Cheetah and Savanna components provide value to applications that are based purely in the S-A-R motif without a need for broader co-design.

Testbeds

The following case studies have been evaluated on a number of different types and scales of platform. The four key platforms whose results we use below are as follows:

Summit: The Summit supercomputer at Oak Ridge National Laboratory is a leadership class machine and the world's fastest supercomputer from June 2018 upto June 2020²³. Each of the approximately 4,600 compute nodes on Summit contains two IBM POWER9 processors and six NVIDIA Volta V100 accelerators, and provides a theoretical double-precision capability of approximately 40 TF. Nodes contain 512 GB of DDR4 memory for use by the POWER9 processors and 96 GB of High Bandwidth Memory (HBM2) for use by the accelerators. Additionally, each node has 1.6TB of non-volatile memory that can be used as a burst buffer. Each of the two POWER9 processors consists of 22 SIMD Multi-Cores (SMCs), where each core supports Simultaneous Multi-Threading (SMT) with up to four hardware threads. Summit nodes are connected to a dual-rail EDR InfiniBand network providing a node injection bandwidth of 23 GB/s. Nodes are interconnected in a Non-blocking Fat Tree topology. This interconnect is a three-level tree implemented by a switch to connect nodes within each cabinet (first level) along with Director switches (second and third level) that connect cabinets together. Summit is connected to an IBM Spectrum Scale filesystem providing 250PB of storage capacity with a peak write speed of 2.5 TB/s.

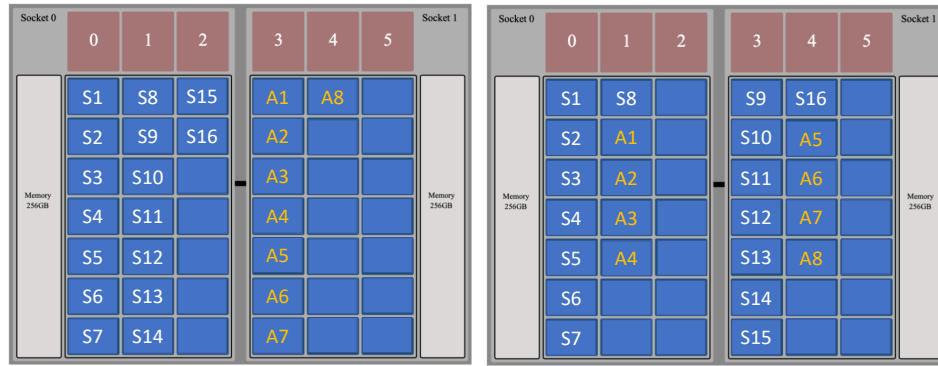


FIGURE 13 Two alternative mappings on a Summit node, in which Simulation (S) and Analysis (A) processes are co-located on separate sockets (left) or the same socket (right).

Rhea: The Rhea system at Oak Ridge National Laboratory is a 521-node commodity-type linux cluster. Its main purpose is to provide a means to perform pre/post processing and analysis of large data. It consists of two partitions, one consisting of CPUs only, and the other consisting of CPUs along with GPUs. The CPU partition consists of 512 nodes, where each node contains two 8-core 2.0 GHz Intel Xeon processors with Intel’s Hyper-Threading (HT) Technology and 128GB of main memory. With hyperthreading enabled, each node has 32 logical cores. The GPU partition has nine large memory GPU nodes. These nodes each have 1TB of main memory and two NVIDIA K80 GPUs in addition to two 14-core 2.30 GHz Intel Xeon processors.

Theta: The Theta supercomputer²⁴ at Argonne National Laboratory is a Cray XC40 system with over 4000 compute nodes. Each compute node contains an Intel Knights Landing 7230 processor with 16 GiB of MCDRAM and 192 GiB of DDR4. Each processor has 64 cores and each core has 4 SMT hardware threads. The machine hosts a 10 PB Lustre parallel file system along with the Aries Dragonfly high speed network interconnect.

Deeptthought2: Deeptthought2²⁵ is a standard Linux cluster at University of Maryland with 448 nodes, where each node has 20 cores (with 2 hardware threads/core) and 128 GB of DDR3 memory running at 1866 MHz. Each node has dual Intel Ivy Bridge E5-2680v2 processors running at 2.80 GHz and the nodes are interconnected with Mellanox FDR Infiniband. Each core has 32KB L1, 256KB L2 as exclusive cache and shares two 25MB of L3 (one per socket) with other cores. It hosts a 1PB Lustre file system with a theoretical maximum throughput of about 56 Gb/s.

4.1 | Determining Efficient Process Placement Strategies for Simulation and Analysis Workflows

In order to demonstrate the power and flexibility of Cheetah to the software engineering challenge of coupled application pipelines, we investigate the particular motif of “online analysis” using a reference application pipeline based upon the Gray-Scott model of reaction-diffusion²⁶ along with an analysis application that calculates the probability distribution of the simulation output. This pipeline is a useful stand-in for a number of similar pipelines since reaction-diffusion equations can generate complex, time-varying patterns.

The applications run concurrently and are coupled in-memory using the SST staging transport in ADIOS. Using the non-blocking mode of SST, the coupling is done asynchronously so that the simulation processes publish data and resume computation without waiting for the readers to read the data. The simulation waits for the analysis to complete before closing the data stream at the end. We focus here on the experimental data gathered from runs on the Summit supercomputer at Oak Ridge National Laboratory, although there are similar demonstrations for other extreme computing platforms.

In our experiments, we explore two categories of process placement and resource distribution options: 1) simulation and analysis processes reside on separate nodes, and 2) processes from both applications share compute nodes. Furthermore, we also explore ‘socket-sharing’, in which simulation and analysis ranks are placed on the same socket of the node. This fine-grained process placement was obtained using Cheetah’s *Virtual Node* interface as described in the previous section. Figure 13 displays configurations in which simulation and analysis ranks are placed on the same compute node of Summit. As shown, the two applications may share sockets or may be placed on separate sockets of the node.

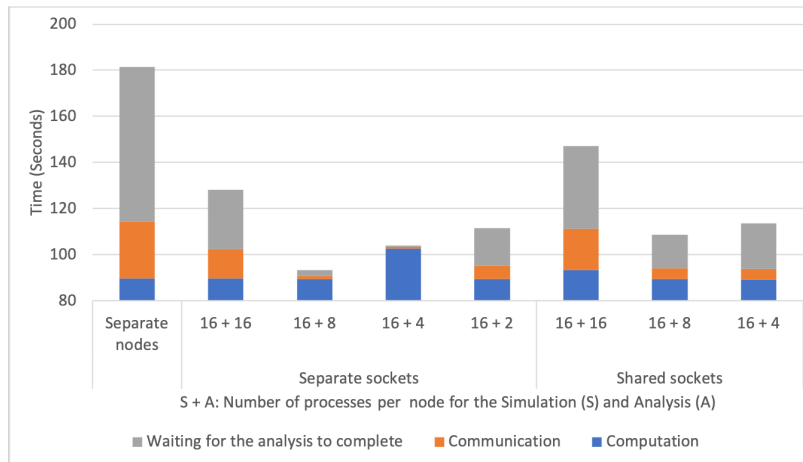


FIGURE 14 Testing the performance of various process placement schemes for the Gray-Scott simulation and analysis pipeline generated using Cheetah on Summit. Overall, we test two categories of process placement mechanisms: 1) simulation and analysis ranks reside on separate nodes, 2) simulation and analysis ranks share compute nodes. The best case of ‘node-sharing’ shows a 45% reduction in runtime as compared to running the two applications on separate nodes.

Each run of the benchmark was executed on 16 nodes of Summit. The total number of simulation ranks was fixed at 256 (16 MPI processes per node), and the number of analysis ranks was varied. The application simulated 50 timesteps, of which 5 were transferred to the analysis application for processing. In each step, 16 GB of data was transferred to the analysis code. More details about the campaign can be found in Mehta et al²⁷. Figure 14 shows the performance of the workflow under different process placement schemes. First, we inspect the ‘separate-nodes’ test case, in which the simulation and analysis applications are placed on separate nodes. A set of experiments was run to determine the best distribution of nodes between them. For brevity, we omit those results, but the best performance was obtained when both the simulation and analysis processes were run on 8 nodes each (256 ranks in total for each application). For the second category of experiments exploring ‘node-sharing’, ranks from both applications shared compute nodes. As each socket contains 21 CPU cores, we spawned 16 simulation ranks per node, and vary the number of analysis ranks on a node/socket. We observe that sharing nodes reduces total workflow runtime significantly, as a majority of the communication is node-local, that is, between processes on the same node. We notice that binding all simulation ranks to a socket and spawning 8 analysis ranks on another socket shows the best performance for the workflow. A detailed analysis of various parameters impacting this behavior is part of our future work. However, the co-design study highlights the impact of fine-grained process placement for in situ pipelines of coupled applications that involve movement and analysis of large volumes of data online.

4.2 | Understanding Aspects of Lossy Compression for In Situ Data Analysis

With the decrease in the ratio of I/O bandwidth to compute-node flops that has been seen over the last decade of computing hardware evolution, it has been increasingly important to understand how to reduce the amount of data being written out of a high performance environment compared to the internal working set. Although some areas have been able to benefit from in situ analysis, visualization, or other online knowledge extraction techniques, there are still cases where it is best to write out as much of the core data as possible. In such situations, having access to high quality lossy compression tools becomes important, as traditional lossless compression cannot guarantee the order of magnitude (or greater) reduction in size for scientific data that is required.

As such, one key application of the co-design study capabilities of Cheetah has been the exploration of how different lossy compression approaches affect the size, fidelity, and reusability of simulation output. This last point is particularly important – reusability for these data sets requires not only that the data fit within particular bounds, but also that key derived features (e.g., isosurface locations, locations of minima and maxima) are also preserved to within a reasonable error bound. In the study we showcase here, we use Cheetah and Savanna to construct a co-design into the trade-offs when selecting the combination of a compression algorithm and its parameters when saving compressed checkpoints of a simulation²⁸. This work determines the trade-off between the compression ratio and preserving the features important for an analysis.

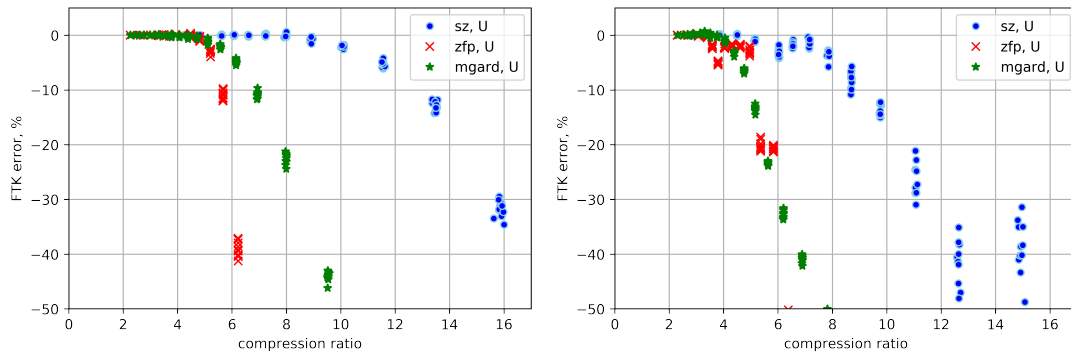


FIGURE 15 The relative error in feature detection versus the compression ratio for the Gray-Scott benchmark after 7000 timesteps (left) and 10,000 timesteps (right).

The experiments shown in Figure 15 were conducted using the Gray-Scott reaction-diffusion benchmark²⁶. The Z-Checker^{29,30} and FTK^{31,32} libraries were used as tools to assess the quality of the data after lossy compression and of the preservation of topological features respectively. Additional feature metrics included ISO surface area, volume, and number of ISO connected components as a measure of feature preservation.

For the purposes of the co-design study, Cheetah was configured for parameter sweeps over 15 values of compression *tolerance*, four values of σ in Gaussian filtering, and three compression algorithms (SZ, ZFP, MGARD). Additionally, 11 trials for each of the $15 \times 4 \times 3 = 180$ combinations were conducted, for a total of $180 \times 11 = 1980$ runs. The campaign of experiments was run on the Summit supercomputer at Oak Ridge National Laboratory and the Theta supercomputer at Argonne National Laboratory.

Figure 15 shows the relative error in the number of local maxima detected by FTK versus the compression ratio for the Gray-Scott benchmark after 7,000 and 10,000 time steps of the simulation. It shows that the optimal choice of the compression algorithm and parameters depends on time, the simulation type, and the feature metrics. The full co-design study looked at additional components, including the run-time impact of choices, the frequency of possible retraining requirements, and the other feature extraction techniques as mentioned above. Those details and more can be found in Yakushin et al²⁸. However, the details of this experiment showcase that the co-design campaigns not only fit well into the Cheetah design space, but there are also important components of the runtime execution environment that might need to be included even in final solutions. The study shows that tuning of the optimal compression algorithm and its parameters should be periodically redone in situ if efficiency and fidelity really are constraints. Using Cheetah and Savanna for such dynamic autotuning and management scenarios is further discussed in section 4.6.

4.3 | EFFIS - A Code Coupling Framework for Exascale Applications

The Exascale Framework for High Fidelity Coupled Simulations (EFFIS)³³ is a workflow and code coupling framework developed as part of the Whole Device Modeling Application (WDMApp) in the Exascale Computing Project. EFFIS consists of libraries, command line utilities, and run-time daemons which together enable users to easily compose and execute complex coupled-application workflows, along with in situ analysis, visualization, and performance monitoring, as well as to deploy services for command-and-control, remote dashboard viewing, post-processing, and more. EFFIS's features are largely motivated in support of WDMApp's objectives, but the software is more generally applicable. It provides a high level state-of-the-art coupling framework that can subscribe to data from one or more science codes and perform the computations and data transformations necessary to the data exchange, analysis, and visualization.

An example EFFIS workflow for the WDMApp pipeline is shown in Figure 16. EFFIS leverages Cheetah and Savanna as its composition and run-time execution back-ends respectively, and provides a custom front-end on top of Cheetah's abstract specification interface. EFFIS's composition layer provides an interface for composing a multi-component, in situ workflow pipeline based upon a combination of pragma-based source code integration and a YAML-based configuration. Its custom front-end generates a Python translation to generate a Cheetah specification. The EFFIS submission command invokes Savanna to execute the workflow pipeline. Savanna equips EFFIS to support all major DOE computing sites, enabling EFFIS development to focus on workflow services, automation, and enhancements. EFFIS provides a set of inbuilt processes such as the dashboard

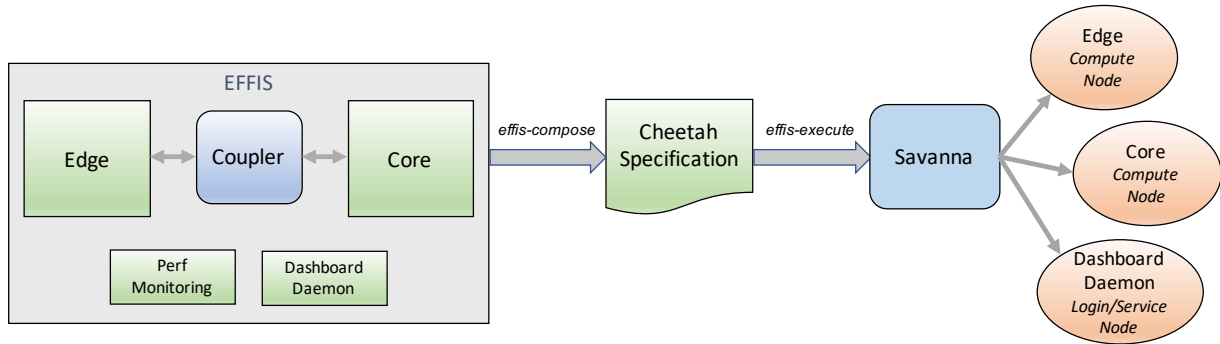


FIGURE 16 An EFFIS workflow showing the Core and Edge codes of the Whole Device Modeling pipeline coupled in memory. The EFFIS user front-end is used to compose an in situ pipeline, which is then translated into a Cheetah specification using an EFFIS compose script. Finally, an execute script in EFFIS launches the Savanna runtime engine for orchestrating and executing the pipeline. The science codes run on a cluster’s computer nodes, whereas the dashboard daemon runs on a login/service node.

daemon, which can be used to export images generated by a simulation to be stored on a publicly accessible server. The Cheetah experiment manifest generated by EFFIS automatically inserts metadata information about EFFIS’s inbuilt processes. At runtime, Savanna automatically executes these daemon processes by selecting the appropriate resources. EFFIS specifically provides the ability to compose and orchestrate multi-tier service stacks, in which simulations are coupled in memory using the ADIOS library and are run on compute nodes of a cluster, whereas service daemons such as performance monitors and dashboard processes are run on login and services nodes of the cluster. Using Cheetah and Savanna greatly simplifies this pipeline orchestration, as the Cheetah interface allows mapping processes to resources such as login and compute nodes. By building upon the Cheetah and Savanna frameworks, EFFIS thus provides a higher-level abstraction with focus on coupling science codes and provides the data transformations required for semantic data exchange between the physics applications of the WDMApp.

4.4 | Co-designing I/O Middleware for Multi-Tier HPC Storage Architectures

Modern HPC facilities contain a multi-level storage hierarchy comprised of device memory, local and/or shared non-volatile memory (NVM), parallel file system (PFS), and tape drives. While additional fast cache layers can help alleviate the pressure of large I/O on the system, effectively utilizing the storage layers can be challenging. This study uses Cheetah and Savanna to conduct a co-design study for HPC I/O middleware for tiered storage architectures. In particular, parameter sweeps are conducted to study effective ways of draining data from fast, local NVM to persistent storage.

Various DOE science applications benefit from this study. The SPECFEM3D_GLOBE³⁴ application is a 3D seismic wave propagation solver that constructs high-resolution tomographic images of the Earth’s mantle at the global scale. In a forward computation phase, it regularly writes large volumes of model data that can consume upto 35% of the total application runtime even on state-of-the-art machines such as the Summit supercomputer at Oak Ridge National Laboratory which hosts a parallel file system with theoretical peak bandwidth of 2.5 Terabytes/second. On the other hand, particle-in-cell codes such as the Gyro-Kinetic Toroidal Code (GTC) write large amounts of checkpoint-restart data periodically that can overwhelm the I/O subsystem.

In this study, we evaluate different ways of writing to the NVM and draining data to persistent storage. A science application writes its large data to the node-local NVM, and daemon processes running on idle cores of the node concurrently drain data from the NVM to the PFS in the background. We study the impact of three parameters on the workflow: n - the number of I/O streams per node that drain the NVM, r - the data rate, or the maximum amount of data being drained in a single drain epoch, and f - the frequency at which the NVM is drained. A higher value of n represents multiple I/O streams flushing data concurrently, thereby utilizing more compute resources per node. Increasing r represents flushing larger blocks of data per epoch, and a higher value of f represents flushing data more frequently. These parameters can be configured to set up infrequent flushing of large data (low f and high r), or continuous, trickle-type draining (high f and low r). Applications with different I/O patterns can configure different types of flushing approaches depending upon the performance characteristics of the application and the underlying system.

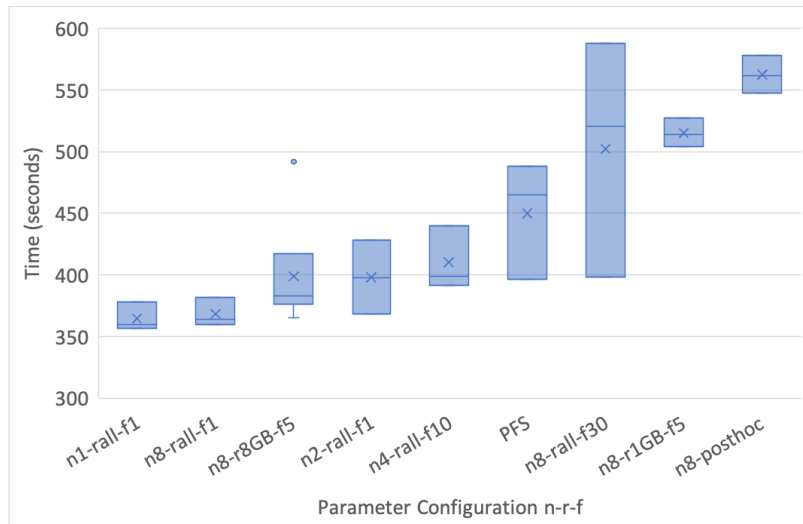


FIGURE 17 The total runtime of the SPECfem3D_GLOBE benchmark for different combinations of n , f , and r for draining the node-local NVM.

We run a campaign on Summit using the SPECfem3D_GLOBE application to study different combinations of n , f , and r . The simulation is run on 800 nodes and generates a total of 260 TB of data in each run. Daemon processes on each node drain data from the NVM to the PFS. Figure 17 shows the impact of different parameter configurations on the total runtime of the application. The different parameter configurations are labelled as n - r - f . For example, $n8$ - $r8GB$ - $f5$ represents 8 daemon processes on each node flushing a maximum of 8 GB of data every 5 seconds. ‘rall’ indicates all available data is flushed from the NVM, and ‘PFS’ represents writing data directly to the parallel file system without using the NVM (base case). ‘posthoc’ represents flushing the NVM only after the simulation is complete - a practice commonly adopted by many supercomputing facilities. As SPECfem3D_GLOBE generates data frequently (2-3 seconds time gap between write epochs), we consider this pattern as high-volume, high-velocity I/O. We see that fast, reactive draining using a single thread performs well, whereas flushing limited amounts of data and at low frequency usually degrades performance. We propose more applications with different I/O patterns can conduct similar co-design studies to study the impact of n , f , and r on using hierarchical storage on different target machines. Using Cheetah helps explore this spectrum of writing data and moving it through the different layers of the hierarchical storage using different techniques.

4.5 | Continuous Performance Testing for Exascale Projects

The Exascale Computing Project (ECP)³⁵ is an initiative geared towards the research, development, deployment, and delivery of mission-critical applications and an integrated software stack for upcoming exascale supercomputers. As part of ECP, the ADIOS framework aims to provide a declarative publish-subscribe input/output interface for managing scientific data at the exascale. ADIOS implements an adaptive, modular approach to providing these generalized I/O services, where a particular application could invoke any of a number of different providers with several different customizable parameters. However, this variety introduces a difficulty in offering good suggestions to an end user about configuration options that can deliver optimal or even adequate performance. As a result, a long-term goal of the ADIOS project has been to develop a repeatable and realistic performance test suite that can be run on pre-exascale and exascale machines to evaluate the performance of I/O bound applications.

For this purpose, the Cheetah and Savanna suite of tools is used to compose and execute a campaign of small- and medium-scale experiments. Parameter sweeps are set up to test different engines and parameters in the ADIOS configuration on a variety of hardware platforms over the variety of different applications and use cases. These include testing the performance of simulations coupled in memory, in situ analysis of data, file I/O, and staging of data.

There are a number of synthetic and application-specific performance examples that have been included in the sweep. As a particular example of how we have put Cheetah and Savanna to use for the testing, consider one of the NWChem computational chemistry workflows shown in Figure 18. Many applications have some computation incorporated that is always, or almost

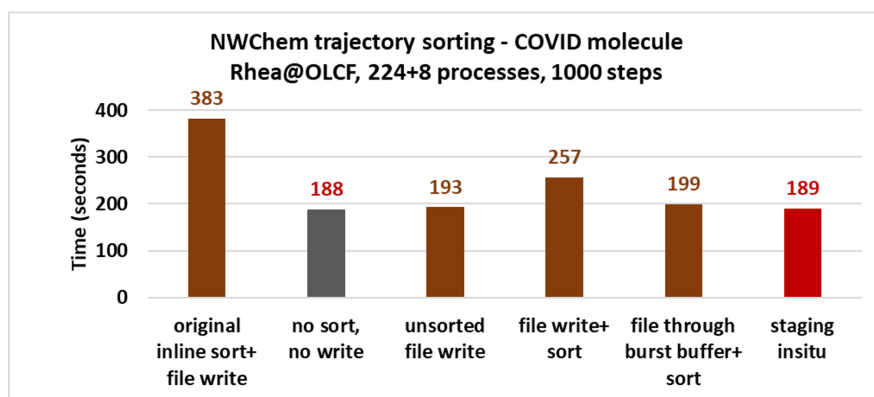


FIGURE 18 ADIOS performance tests of the NWChem trajectory sorting pipeline, comparing the original code with sorting the atom coordinates in a separate program. The unsorted data is moved either through files, or through files that are asynchronously drained to the file system through a burst buffer, or directly to the sorter’s memory using an ADIOS staging engine.

always, necessary for any post-processing work but that does not scale as well as the application itself, and so it becomes a bottleneck over time as users run larger and longer simulations with these applications. Separating the non-scaling, costly computation from the application and performing it in situ or online may bring down the overall cost of the operation while still providing the result by the end of the simulation. The difficulty is that it is not obvious where the operation should be placed and how the data should be moved between the application and the operation.

NWChem is one such simulation, where the wandering of atoms in space (and across simulation processes) requires a distributed sorting method. The first and second columns in Figure 18 show that sorting and writing the atom coordinates in the application itself is over half of the entire simulation run, a costly IO overhead. Using ADIOS to just dump the unsorted data to the file system has very low overhead (third column) but if a second application (the sorter) is continuously reading the same file, interference through the file system operation slows down the simulation (fourth column). If the data is written to a burst buffer and then drained asynchronously to the file system, interference is observed between the draining threads and the sorter, and the slowdown in the simulation is reduced (fifth column). However, the best way of sorting the atoms (on this particular machine and simulation case) is to use memory-to-memory data staging from the simulation to the sorter without using the file system (last column).

Enabling Portability and Reusability: Beyond the goal of using such tests to give feedback to end users as part of initial performance tuning experiments, the reliable reusability of the testing campaigns once they have been put into Cheetah offers a particular benefit for performance engineering during code development. Just as one can do continuous integration (CI) for correctness of new code additions, the goal of the Cheetah campaigns for ADIOS is to enable regular performance evaluations of the impact of code additions. Because of the scale at which such tests need to run, these aren’t intended to run as often as a CI would, but they are intended to run frequently. Furthermore, they also allow testing of different topology and process placement strategies for application processes on different hardware.

Integrating Services: This capability for enabling regular development-focused performance testing raises the question of how to relay such feedback back to the developer, or indeed the community at large. One interesting feature of Cheetah and Savanna that this highlights is the importance of considering the entire workflow life cycle of the data that is collected through the campaign. Cheetah’s ability to automatically append additional daemon processes or other helper services to a campaign without requiring additional heavy lifting for the end user is particularly useful in scenarios like this. Persistent testing results, when done right, will need to be gathered from a variety of machines and configurations. Additionally, keeping time traces of performance to aid in debugging and understanding performance trends will be important.

The ADIOS team has worked with Kitware, Inc. to implement a performance tracking dashboard in conjunction with the implementation of the campaign specifications in Cheetah, a snapshot view of which can be seen in Figure 19. The needed services to process the immediate output from the performance test are included through the post-sweep hooks in Cheetah, and the services can be configured depending on the situation for the data to be sent to a pre-existing dashboard, or for the dashboard service process to be launched and managed by Savanna itself, as needed.

When coupled to a persistent third party service, the automatic daemon collection process(es) can enable both immediate and time evolution views on the performance data. The example in Figure 20 shows an early version of this interface. This is

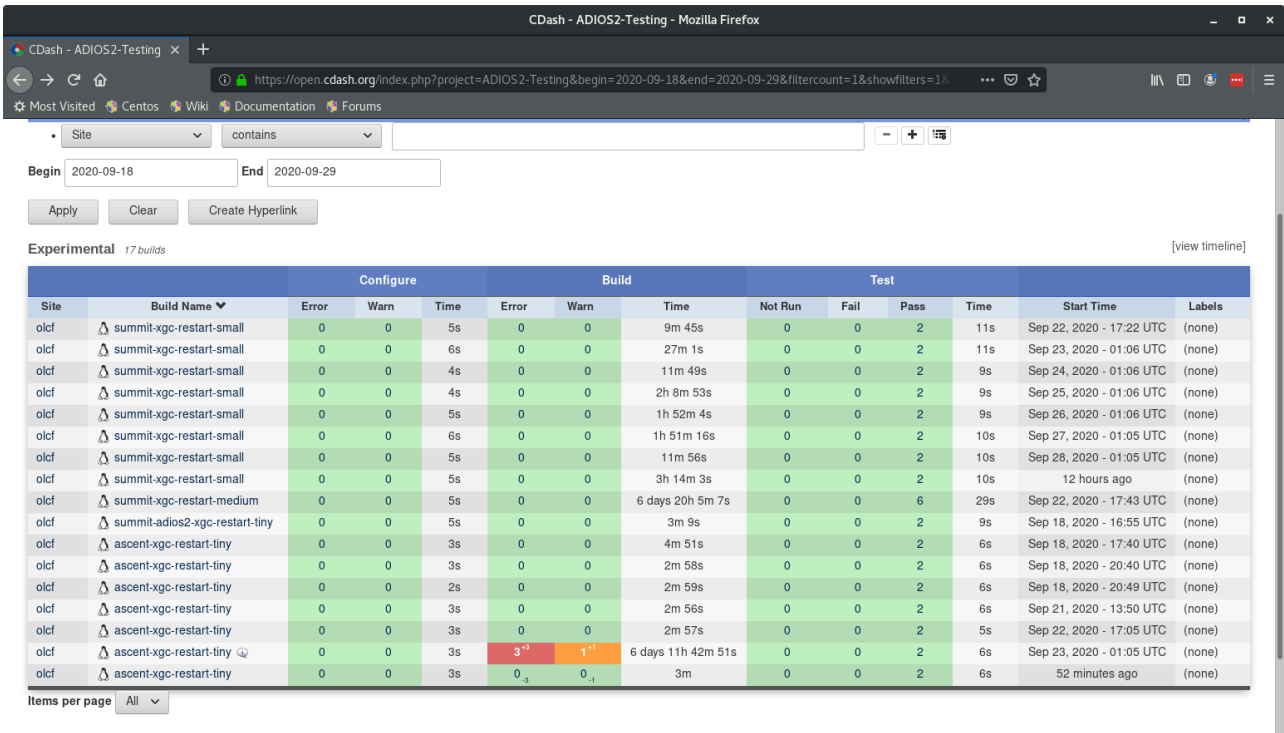


FIGURE 19 The dashboard showing a summary of the different sweep groups run on different machines as part of the ADIOS performance testing suite created using Cheetah.

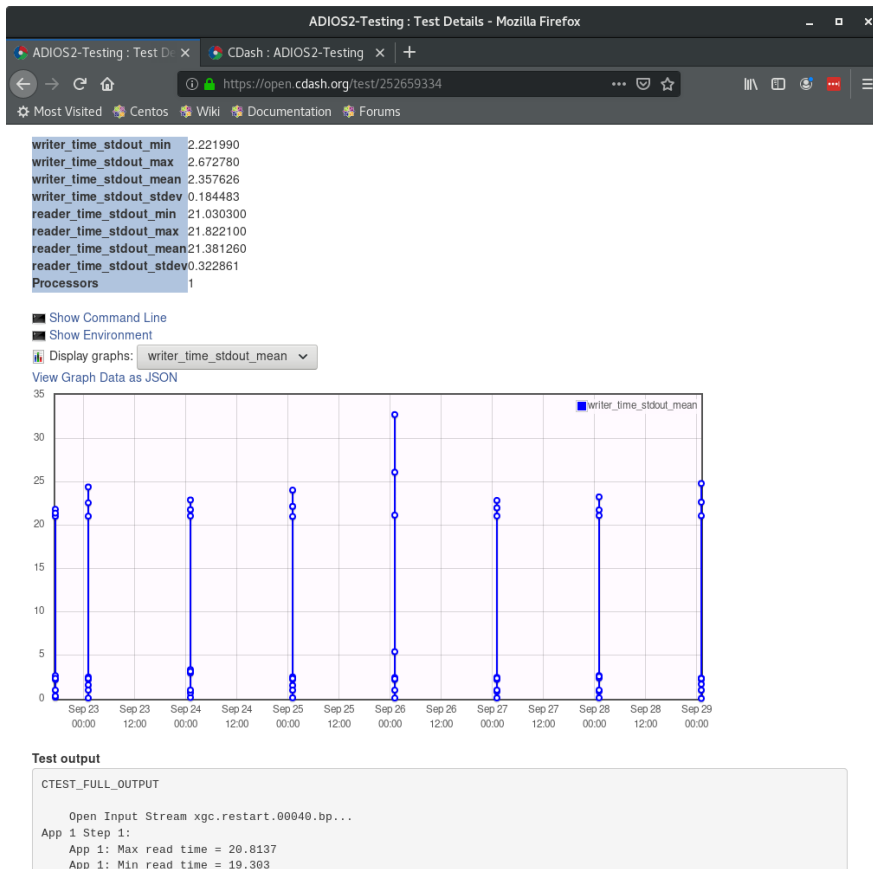


FIGURE 20 The dashboard showing a sample of the performance statistics displayed for a particular sweep. The dashboard is deployed as part of the performance testing suite created using Cheetah.

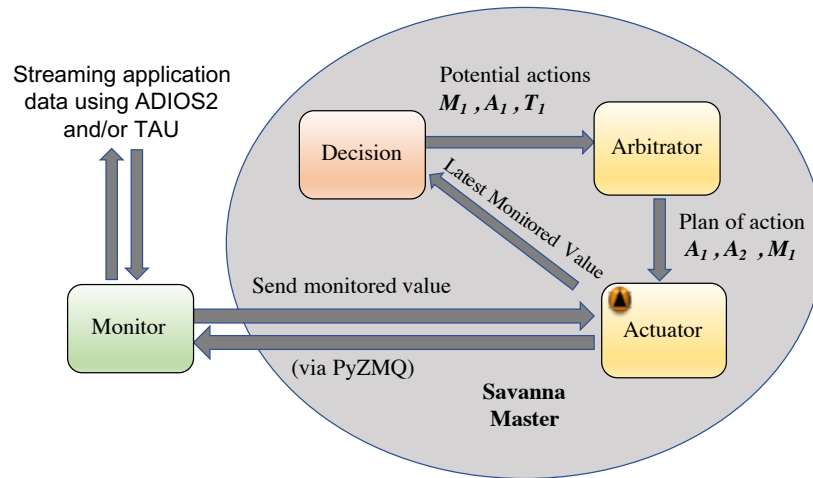


FIGURE 21 Extensions performed on the Savanna master to enable dynamic control for orchestrating workflows at runtime.

particularly important when considering performance tracking experiments on HPC facilities where the performance testing codes will be competing with other job submissions for network and I/O bandwidth. A poor result may be due to a newly introduced performance bug, or it may be that another user just swamped the file system. By using Cheetah's ability to track not only the sweeps but also a repetition count on particular experiments, one can build more complex profiles of the expected performance impact, and thus allow building better models for workflow performance on shared supercomputer systems.

4.6 | Dynamic Strategies for Managing Scientific Workflows

Because of advances in computational power and new technologies enabling online data analysis, modern scientific workflows are growing in size and complexity. Unlike more traditional workflows, the runtime behavior of these workflows is more unpredictable, and a fixed, predetermined (static) resource assignment can be inefficient for overall performance. Therefore, a need for autonomous dynamic workflow controls is emerging that can adapt resource assignments according to the changing runtime needs of workflows. This can enable applications and their users to explore innovative ways to improve overall performance and best utilize available computational resources.

Such dynamic control requires four aspects in a workflow toolset - *Monitoring*, *Decision Making*, *Arbitration* and *Actuation*. The Monitoring component continuously gathers information needed from the running workflow to assess if an event of interest has occurred. The decision making aspect assesses information and selects a policy from a list of available policies to actuate a dynamic action. The Arbitrator and Actuator functions construct and execute a plan of action to implement the selected policy and ensure consistency with the workflow specifications.

To implement such dynamic control, we extended Cheetah and Savanna as shown in Figure 21 to implement the components described above. The Monitor function directly communicates with the workflow applications. It is a hybrid MPI and Python-based multi-threaded tool that is transparently launched by Savanna. The Monitor continuously receives profiling information about the workflow components and shares it with Savanna (the Actuator) using the PyZMQ³⁶ messaging service. The Decision function is invoked by the Actuator based on a trigger protocol. Once a set of potential actions are identified by the Decision function, the Arbitrator function is invoked which then constructs the final plan of action. The Actuator applies these actions to the workflow components. Examples of monitoring information include a workflow component's timestep throughput or communication overhead. Simple rule-based policies allow users to stop and restart an application with different resource allocations (e.g., processor cores, memory capacity, network bandwidth) using threshold-based conditions. The Cheetah input specification format was extended to allow users to insert monitoring functions and a set of rules to apply at runtime.

With the above extensions, we observe promising results across different workflows and architectures. Figure 22 shows one such example from a LAMMPS molecular dynamics workflow where an under-provisioning situation was corrected at runtime. Here, LAMMPS was coupled with analysis components that compute a radial distribution function (RDF_Ca1c), perform a common neighbor analysis (CNA_Ca1c), and compute central symmetry (CS_Ca1c). In the initial resource assignment, LAMMPS was started with 1000 processes (10 per node) to complete 1000 iterations. All three analysis applications, i.e. CS_Ca1c,

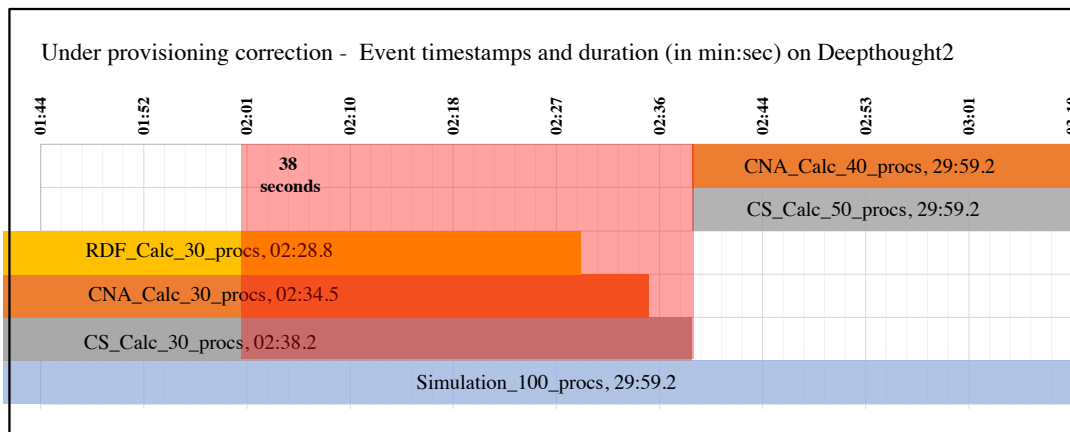


FIGURE 22 An experiment that demonstrates the use of dynamic controls implemented as extensions to the Cheetah-Savanna suite for correcting under-provisioning of a workflow from a molecular dynamics workflow. The CNA and CS calculations are re-configured dynamically based upon a dynamic profile of the workflow.

CNA_Calc and RDF_Calc, were started concurrently with 30 processes each (3 per node). The experiment was allocated a maximum of 30 minutes. The workflow was configured to monitor the throughput represented as the number of timesteps computed in a set period of time. During the course of the experiment, the CS_Calc and CNA_Calc applications were observed to exhibit low throughput due to insufficient compute resources allocated to them. The dynamic control framework actuated a load-rebalancing policy which reassigned cores from the RDF_Calc application to the CS_Calc and CNA_Calc components. The reconfiguration took less 2% of the execution time of the entire workflow.

We note that Cheetah’s declarative-style interface provides a more natural way to express dynamism in a workflow. As part of future work, we aim to further extend dynamic capabilities in the campaign specification.

5 | RELATED WORK

The history of using workflow tools to support scientific computation is long³⁷. Entire books have been filled with details of important efforts; we focus here on some of the specific efforts and techniques that have influenced our decisions. ZOO³⁸, Zenturio³⁹, Nimrod⁴⁰, Emulab⁴¹, Expertus⁴², Pegasus⁴³, and perfbase⁴⁴ are examples of systems designed to run and collect results from sets of computational experiments. They differ in how experiments are defined (e.g., via a language in Zenturio⁴⁵, templates in EmuLab⁴¹), the types of experiments supported (e.g., performance experiments in Zenturio, Emulab, and perfbase; computational experiments in Nimrod and ZOO), and the types of computations supported (e.g., high-performance computations in perfbase, network experiments in EmuLab, clouds for Expertus).

Various job schedulers and resource management tools address job submission and management for large machines. Balsam⁴⁶ is a task-based job scheduling tool that focuses on improving job throughput on supercomputers. Swift/T¹⁸ provides a bag of tasks model to run dataflow-based workflows on large machines. Flux¹⁷ is a resource management framework that aims to provide a consistent interface across different machines. It provides a Python-based API that allows fine-grained control of job management. Parsl¹⁹ is a dataflow-based workflow tool and scripting library. Its libSubmit library provides a scheduler abstraction for submitting jobs. RADICAL-Pilot²⁰ is another Python-based job management system that aims to obtain high job throughput. An extension of it is the RADICAL-Ensemble Toolkit⁴⁷ for building and running ensemble workflows.

In comparison, Cheetah and Savanna are designed around the *Campaign* model of execution. The focus is on composing and executing a workflow of workflows, as opposed to efficiently building and running a single workflow. A new workflow pattern termed ‘Bundled Workflows’ provides the capability to run multiple, distinct applications concurrently as if they are part of a single application. Resources are provisioned so that all applications that are part of a bundle can be launched concurrently. The VirtualNode interface in Cheetah-Savanna provides a novel fine-grained process placement abstraction for exploring different

topologies. The Cheetah-Savanna toolset provides an *end-to-end* solution from campaign composition, execution, tracking, to performance introspection.

6 | SUMMARY AND FUTURE WORK

Simulation-Analysis-Reduction pipelines represent a promising new programming paradigm for scientific codes, coming out of the Online Data Analysis and Reduction motif. In this paper, we have discussed the motifs and challenges behind the co-design of scientific S-A-R application pipelines. We have illustrated the challenges involved in conducting co-design studies with in-memory coupled workflows, and have demonstrated the need for a sophisticated toolset for conducting such studies. We have introduced the Cheetah and Savanna toolset for conducting co-design studies. Cheetah-Savanna are centered around the campaign model which represents a workflow of workflows. Cheetah is a co-design experiment harness that provides a high-level, Python-based specification for co-design campaigns, whereas Savanna is an online runtime framework for orchestrating and executing workflows. We have introduced the Virtual Node object model for fine-grained process placement and workflow orchestration. We demonstrate the advantage of using Cheetah through various use cases that explore complex process placement, compression studies, dynamic workflow controls, and a performance testing campaign suite for an ECP (Exascale Computing Project) project. Experiments were conducted on pre-exascale leadership machines such as the Summit supercomputer at Oak Ridge National Laboratory and the Theta machine at Argonne National Laboratory.

To continue building a community of users around our tool base, we want to explore opportunities in understanding new motifs that may mix the online data analysis approach with other workflow models such as those supported by tools such as Parsl, Flux, and Radical. Looking toward future hardware, there is a need to support an even wider range of configuration options (e.g., in situ as well as in transit) and architectural features (e.g., accelerators, NVRAM, NVMeoverFabric). Similarly, we want to develop and implement local shared-memory based communication methods when a node is shared between applications, to make sure we extract the best possible performance in those scenarios.

In a more general frame, we want to explore appropriate use of design of experiment methods⁴⁸ and optimization-based orchestration^{49,50} to minimize the number of experiments required to answer a co-design question. The software development process needs tools to help guide efficiency of the delivered Simulation-Analysis-Reduction scientific codes, but those tools themselves can benefit from being efficient in how they generate their results. Our goal is for Cheetah and Savanna to serve as bedrock tools for the development of the next-generation ecosystem of complex, multi-application scientific computing. Through co-design performance and capability optimization, Cheetah guides developers to fast, lean, and portable solutions for their pipelines, while Savanna offers the open ecosystem runtime for the high performance ODAR motif.

ACKNOWLEDGEMENT

This research was supported in part by the Exascale Computing Project (17-SC-20-SC) of the U.S. Department of Energy (DOE), and by DOE's Advanced Scientific Research Office (ASCR) under contract DE-AC02-06CH11357. Additionally, this research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory and of the National Energy Research Scientific Computing Center, which are supported by the Office of Science of the U.S. Department of Energy under Contract Numbers DE-AC05-00OR22725 and DE-AC02-05CH11231, respectively.

7 | BIBLIOGRAPHY

References

1. Hamilton S, Burns R, Meneveau C, et al. Extreme event analysis in next generation simulation architectures. In: *International Supercomputing Conference*; 2017: 277–293.
2. Foster I, Ainsworth M, Bessac J, et al. Online Data Analysis and Reduction: An Important Co-design Motif for Extreme-Scale Computers. *International Journal of High-Performance Computing Applications* 2020; in press.

3. Foster I, Ainsworth M, Allen B, et al. Computing just what you need: Online data analysis and reduction at extreme scales. In: *Euro-Par: The European Conference on Parallel Processing*; 2017.
4. Ang J, Brightwell R, Donofrio D, et al. Exascale Computing and the Role of Co-design. In: *High Performance Computing: From Grids and Clouds to Exascale*IoS Press. 2011 (pp. 43–64).
5. Choi JY, Chang C, Dominski J, et al. Coupling Exascale Multiphysics Applications: Methods and Lessons Learned. In: *2018 IEEE 14th International Conference on e-Science (e-Science)*; 2018: 442–452
6. Liu Q, Logan J, Tian Y, et al. Hello ADIOS: the challenges and lessons of developing leadership class I/O frameworks. *Concurrency and Computation: Practice and Experience* 2014; 26(7): 1453–1473. doi: 10.1002/cpe.3125
7. Shende SS, Malony AD. The Tau Parallel Performance System. *Int. J. High Perform. Comput. Appl.* 2006; 20(2): 287–311. doi: 10.1177/1094342006064482
8. Lindstrom P. Fixed-Rate Compressed Floating-Point Arrays. *IEEE Transactions on Visualization and Computer Graphics* 2014; 20(12): 2674–2683.
9. Di S, Cappello F. Fast Error-bounded Lossy HPC Data Compression with SZ. In: *IEEE International Parallel and Distributed Processing Symposium*; 2016: 730–739.
10. Ainsworth M, Tugluk O, Whitney B, Klasky S. MGARD: A multilevel technique for compression of floating-point data. In: *DRBSD-2 Workshop at Supercomputing*; 2017.
11. Ainsworth M, Tugluk O, Whitney B, Klasky S. Multilevel Techniques for Compression and Reduction of Scientific Data—The Unstructured Case. *SIAM Journal on Scientific Computing* 2020; 42(2): A1402–A1427.
12. LZ4—Extremely fast compression. <https://github.com/lz4/lz4>; 2020.
13. Logan J, Mehta K, Heber G, et al. A Vision for Managing Extreme-Scale Data Hoards. In: *IEEE International Conference on Distributed Computing Systems (ICDCS)*; 2019.
14. Jette MA, Yoo AB, Grondona M. 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: 44–60.
15. Tang W, Lan Z, Desai N, Buettner D. Fault-aware, utility-based job scheduling on Blue Gene/P systems. In: *2009 IEEE International Conference on Cluster Computing and Workshops*; 2009: 1–10
16. Shende SS, Malony AD. The TAU parallel performance system. *The International Journal of High Performance Computing Applications* 2006; 20(2): 287–311.
17. Ahn DH, Garlick J, Grondona M, Lipari D, Springmeyer B, Schulz M. Flux: A Next-Generation Resource Management Framework for Large HPC Centers. In: *2014 43rd International Conference on Parallel Processing Workshops*; 2014: 9–17
18. Wilde M, Hategan M, Wozniak JM, Clifford B, Katz DS, Foster I. Swift: A language for distributed parallel scripting. *Parallel Computing* 2011; 37(9): 633–652.
19. Babuji Y, Woodard A, Li Z, et al. Parsl: Pervasive Parallel Programming in Python. In: *HPDC'2019*; 2019.
20. Merzky A, Turilli M, Maldonado M, Santcroos M, Jha S. Using Pilot Systems to Execute Many Task Workloads on Supercomputers. In: Klusáček D, Cirne W, Desai N., eds. *Job Scheduling Strategies for Parallel Processing*Springer International Publishing; 2019; Cham: 61–82.
21. Summit User Guide. https://docs.olcf.ornl.gov/systems/summit_user_guide.html; 2020.
22. Explicit Resource File (ERF) format. https://www.ibm.com/support/knowledgecenter/en/SSWRJV_10.1.0/jsm/erf_format.html; 2020.
23. TOP500. <https://www.top500.org/lists/top500/>; 2020.

24. Theta Machine Overview | Argonne Leadership Computing Facility. <https://www.alcf.anl.gov/support-center/theta/theta-thetagpu-overview>; 2020.
25. Deeptthought2 HPC Cluster. <https://hpcc.umd.edu/hpcc/dt2.html>; .
26. Pearson JE. Complex patterns in a simple system. *Science* 1993; 261(5118): 189–192.
27. Mehta K, Allen B, Wolf M, et al. A Codesign Framework for Online Data Analysis and Reduction. In: *2019 IEEE/ACM Workflows in Support of Large-Scale Science (WORKS)*; 2019: 11–20.
28. Yakushin I, Mehta K, Chen J, et al. Feature-Preserving Lossy Compression for In Situ Data Analysis. In: *49th International Conference on Parallel Processing Workshops* Association for Computing Machinery; 2020; New York, NY, USA
29. Tao D, Di S, Guo H, Chen Z, Cappello F. Z-checker: A framework for assessing lossy compression of scientific data. *International Journal of High Performance Computing Applications* 2019; 33(2): 285–303. doi: 10.1177/1094342017737147
30. Z-Checker repo. <https://github.com/CODARcode/Z-checker>; .
31. Guo H, Lenz D, Xu J, et al. FTK: A High-Dimensional Simplicial Meshing Framework for Robust and Scalable Feature Tracking. *Transactions on Visualization and Computer Graphics* 2021.
32. FTK repo. <https://github.com/CODARcode/ftk>; .
33. Suchyta E, Klasky S, Podhorszki N, et al. The Exascale Framework for High Fidelity coupled Simulations (EFFIS): Enabling whole device modeling in fusion science. *International Journal of High Performance Computing Applications* 2021. doi: 10.1177/109434202111019119
34. Lefebvre M, Chen Y, Lei W, et al. *Data and workflow management for exascale global adjoint tomography*: 279–306; CRC Press . 2017
35. Messina P. The exascale computing project. *Computing in Science & Engineering* 2017; 19(3): 63–67.
36. <https://zeromq.org/languages/python/>; 2019.
37. Existing Workflow Systems. <https://s.apache.org/existing-workflow-systems>; 2020.
38. Ioannidis YE, Livny M, Ailamaki A, Narayanan A, Therber A. ZOO: A desktop experiment management environment. *ACM SIGMOD Record* 1997; 26(2): 580–583.
39. Prodan R, Fahringer T. Zenturio: An experiment management system for cluster and grid computing. In: *IEEE International Conference on Cluster Computing*; 2002: 9–18.
40. Abramson D, Sosic R, Foster I, Giddy J, Lewis A, White N. The Nimrod computational workbench: A case study in desktop metacomputing. In: *Australian Computer Science Conference*; 1997.
41. Eide E, Stoller L, Lepreau J. An Experimentation Workbench for Replayable Networking Research. In: *4th USENIX Symposium on Networked Systems Design & Implementation*; 2007.
42. Jayasinghe D, Swint G, Malkowski S, et al. Expertus: A generator approach to automate performance testing in IaaS clouds. In: *5th International Conference on Cloud Computing*; 2012: 115–122.
43. Deelman E, Singh G, Su MH, et al. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming* 2005; 13(3): 219–237.
44. Worringer J. Experiment management and analysis with perfbase. In: *IEEE International Conference on Cluster Computing*; 2005: 1–11.
45. Prodan R, Fahringer T. ZEN: A directive-based language for automatic experiment management of distributed and parallel programs. In: *International Conference on Parallel Processing*; 2002: 93–100.

46. Salim M, Uram T, Childers J, Balaprakash P, Vishwanath V, Papka M. 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.
47. Balasubramanian V, Turilli M, Hu W, et al. Harnessing the power of many: Extensible toolkit for scalable ensemble applications. In: *2018 IEEE international parallel and distributed processing symposium (IPDPS)*; 2018: 536–545.
48. Montgomery DC. *Design and Analysis of Experiments*. John Wiley & Sons . 2017.
49. Malakar P, Vishwanath V, Munson T, et al. Optimal scheduling of in-situ analysis for large-scale scientific simulations. In: *SC15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*; 2015: 1–11.
50. Malakar P, Vishwanath V, Knight C, Munson T, Papka ME. Optimal execution of co-analysis for large-scale molecular dynamics simulations. In: *SC16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*; 2016: 702–715.

How to cite this article: Mehta, K., B. Allen, M. Wolf, J. Logan, E. Suchyta, S. Singhal, J. Choi, K. Takahashi, K. Huck, I. Yakushin, A. Sussman, T. Munson, I. Foster, and S. Klasky (2021), A Co-design Framework for Online Data Analysis and Reduction, *Concurrency Computat Pract Exper*, 2021;00:x–y.