SANDIA REPORT
SAND2020-9451
Printed August 2020

Sandia National Laboratories

Data Services for Visualization and Analysis – ASC Level II Milestone (7186)

Gary Templet, Matthew Glickman, Todd Kordenbrock, Scott Levy, Jay Lofstead, Jeff Mauldin, Thomas Otahal, Craig Ulmer, Patrick Widener, and Ron Oldfield

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods

ABSTRACT

A new in transit Data Service is presented and compared to the traditional *file-based* workflow and the newly refactored in situ Catalyst workflow. Each workflow is enabled by the IOSS mesh interface equipped with data management layers for Exodus and CGNS (*file-based*), Catalyst (in situ), and FAODEL (in transit). FAODEL is a distributed object store that can transmit data across MPI allocations. Catalyst is a ParaView-based visualization capability developed as part of the CSSE Data Services effort. The workflows considered here take SPARC data into Catalyst for visualization post-processing. Although still in unoptimized form, we show that the in transit approach is a viable alternative to *file-based* and *in situ* workflows and offers several advantages to both simulation and post-processing developers. Since IOSS is a mature interface with wide adoption across Sandia and externally, each workflow can be reconfigured to use different simulations that generate mesh data and post-processing tools that consume it.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

Introduction

The overall goal of this work was to 1) demonstrate *in transit* workflows 2) promote IOSS as a performance portability layer 3) lay out a roadmap for development environments that support workflows and 4) rally the CSSE I/O and Visualization teams around workflow development. These goals further the Data Services approach to HPC systems by fostering development of new workflows that are responsive to users and ultimately deployable in NGW and other workflow management tools.

Milestone Description and Completion Criteria

As displayed in the ASC Implementation Plan (IP) and the Milestone Reporting Tool (MRT), the milestone description and completion criteria state:

1. An initial release of the Catalyst data service utilizing the FAODEL data management layer.
2. A performance and resource-usage evaluation of *in situ*, file system-based workflow, and the data-service approaches.

Impact Statement

There are two primary impacts of this milestone: a practical example to assess of the viability of data services for ASC applications, and an understanding of gaps and potential areas for improvement to our computing environment to better support this model of computing for production on ASC platforms. Lessons learned from this milestone will drive future work in our CSSE path forward in both our I/O and Data Analysis portions of the program.

Summary of Work Done

Note that all results, chapter/section, and table/figure numbers below are taken from SAND2020-9451.

The first goal and exit criteria 1 and 2 were completed and documented in Sec.6.4 which provides workflow performance and resource usage analysis from experiments conducted using 64 nodes. Table 6-1 and Figure 6-9 provide raw timing and memory usage data for each workflow. Each

workflow was also executed in SAW (figures 7-1,7-2,7-3). These results show workflow tradeoffs: *file based* workflows tax I/O bandwidth and disk space but are a "tried and true" option; *in situ* greatly reduces I/O bandwidth used but increases CPU load and development complexity; *in transit* increases overall complexity but does so by offloading data management duties from simulation developers into a dedicated data management layer. *in transit* potentially provides the most flexibility considering the use of workflow management tools as a means of configuring workflows with multiple data producers and consumers.

A more detailed discussion of tradeoffs for each workflow from a performance perspective is found in Sec.6.4. Further discussion in Sec.7.5 describes the impact workflows might have on simulation development costs. The current *in transit* workflow exhibits a software defect when handling Exodus data but works as expected for CGNS data. The build/run process for these workflows and experiments are documented in Appendix A. This is the handoff criteria until the current work can be made production ready with software fixes and optimizations to the FAODEL data management layer Sec.6.1.

The second goal and exit criterion 2 were also demonstrated in Sec.6.4 using IOSS as the data management layer for each workflow. This minimized the development costs for SPARC as it already uses IOSS for I/O (Sec.7.2) Changes to SPARC for this work enabled new IOSS data management layers as first-class options in SPARC's configuration files.

The third goal is demonstrated in Sec.8.1 where lessons learned from this work drive recommendations for changing HPC systems so that they support workflows more easily. Issues include Dynamic Resource Management, Security Models Designed to Support Data Services, and Containerized Workflows for *in situ* visualization.

The fourth goal was demonstrated by the important contributions of the Catalyst team to the L2 work, spanning all workflows (Sec.4.3,4.4,4.1). Also, the Catalyst team accelerated their development schedule to help make IOSS a common tool in each workflow also impacting the second goal. They also provided insight in debugging *in transit* workflows and advice on building and running SPARC.

Path Forward

As mentioned above, successful completion of this effort has given Sandia a demonstration of an *in transit* workflow that couples SPARC and Catalyst. The use of IOSS as a vehicle for engaging with ASC integrated codes, evaluating research ideas in I/O and data analysis, and as a tool for rapid deployment of our technology is now a core part of our R&D strategy moving forward. As mentioned in Chapter 8 of the report, this milestone identified a number of research ideas to improve support for data-services in our HPC computing environment. We are hoping to aggressively pursue some of these potentially disruptive ideas as part of our CSSE research plan, but also through other means (e.g. Office of Science, LDRD, etc.). The target vehicle for *in transit* and the refactored Catalyst *in situ* workflows is NGW as described in Section 8.3.1. Once the *in transit* workflow described in this work is complete and optimized, NGW could deliver *in transit* to a wide variety of IOSS based computational tools.

ACKNOWLEDGEMENTS

We thank the SPARC team, who supported this project with guidance to understanding SPARC and insight into how workflows could impact their work and development cycles. We also thank Gavin Baker who provided insights into resource and performance issues early on in this project. Finally we also thank the review committee for this milestone who patiently offered advice and guidance throughout this process.

1. MILESTONE STATEMENT & EXIT CRITERIA

This work addresses the following ASC Level II milestone:

> As the volume and complexity of data being generated by ASC applications continues to increase, the need to develop technologies to couple simulation and analysis becomes more necessary. For nearly a decade, research projects have been developing *in situ* visualization capabilities like ParaView/Catalyst that combine (at compile time) a scientific simulation with visualization software to provide analysis directly on the application data structures. Other efforts like the Sandia Analysis Workbench (SAW) loosely couple simulation with analysis through shared access to a parallel file system. We are proposing a third approach, a Catalyst data service that couples analysis and simulation through a middle-tier FAODEL data-management layer. While the *in situ* approach has potential to provide the highest-resolution analysis, it introduces a number of practical challenges that make it extremely difficult and hinder productivity of the computational scientist. First, analysis and visualization algorithms often require significant memory and processing resources that large-scale applications may want/need. Second, linking a fairly complex general-purpose library like Catalyst is extremely complicated and potentially fragile. Both the application and the library may use the same third-party libraries and if they happen to use different versions, getting the correct compiling configuration is tedious and problematic. A number of other issues exist that we will expand on in the text of the milestone document.
>
> Using a workflow tool like SAW is also acceptable, but now, instead of having direct access to the data structures, the application and the analysis code use a parallel file system where data is written, then read using a common file format (e.g., Exodus, CGNS). Tools like SAW provide an "automation" capability to couple simulation with analysis. Using a file system as a communication mechanism between application and simulation should be portable and avoids some of the issues identified with *in situ* approaches, but is limited by capacity and bandwidth of the file system and puts a significant burden on the metadata management system of a file system. The data-service approach is a hybrid of the tightly-coupled in situ approach and a loosely coupled SAW workflow. The service executes as a separate job in the HPC system to avoid some of the practical issues of in situ analysis, and it accesses an in-system data-management layer, FAODEL, an alternative to file systems designed to support application workflows. This work will also demonstrate how to couple components in a SAW workflow using the FAODEL data-management layer instead of a file system.
>
> This milestone includes development of a ParaView/Catalyst data service, selection of a representative application (preferably from ATDM or Integrated codes),

and a performance and resource-usage evaluation of *in situ*, file-based workflow, and data-service approaches. We expect the evaluation will require almost no changes to the application source code since we are planning to implement the Catalyst data service as a new backend to the IOSS library, already in use by ATDM and IC applications.

1.1. Satisfaction of Exit Criteria

This report presents details of the satisfaction of the exit criteria for this milestone. In particular:

- **An initial release of the Catalyst data service utilizing the FAODEL data management layer.**

  This initial release is detailed in Appendix A. Each source code repository used in creating the SPARC/Catalyst workflows described in this report is listed, along with the unique identifiers (git commit hash and associated tag) which can be used to retrieve the source code associated with completion of this milestone.

  This release includes software changes to IOSS which enable each workflow to engage the different IOSS data management layers. Also included are instructions for building the entire SPARC/Catalyst/FAODEL software stack on mutrino for KNL and Haswell nodes, and running the HIFiRE-1 example in SPARC for each workflow.

  The Catalyst data service runs correctly with CGNS mesh data. A software defect when handling Exodus mesh data remains to be resolved before the associated code can be committed to the master IOSS repository. Also, IOSS and FAODEL code changes must go through formal release processes before being pushed to the public repositories for IOSS (`https://github.com/gsjaardema/seacas` and FAODEL `https://github.com/faodel/faodel`.

- **A performance and resource-usage evaluation of *in situ*, file system, and data-service workflow approaches.**

  Our performance and resource usage evaluations are summarized in Section 6.4, including a discussion of the tradeoffs posed by the different workflow composition strategies. Details are presented throughout Chapter 6 and include performance and resource management results derived from experiments based on a 32-rank SPARC run. Total runtimes for each workflow are presented, as well as a breakdown of memory usage and runtime for individual workflow components.

  We also describe the interactions our work has had with the code bases and products we sought to integrate, including IOSS, SPARC, and the Sandia Analysis Workbench (Chapter 7). Our evaluation concludes with a discussion (Chapter 8) on how our work might contribute to a future data services ecosystem at Sandia.

2. EXPERIMENT DRIVER

We demonstrated three workflows (depicted in Figure 2-1) that provide a data conduit between SPARC and Catalyst. For each workflow, SPARC computes air flow over the HIFiRE-1 (Hypersonic International Flight Research and Experimentation) vehicle [25]. HIFiRE-1 is an aero vehicle that was used in experiments to gather high-resolution, full-scale heating and pressure data. HIFiRE-1 data is used extensively to validate CFD codes. It is also of research interest to SNL staff studying how to build reduced order models (ROMs) using data from large numbers of small scale (~4 nodes) SPARC runs, as well as staff investigating algorithms for computing flow statistics. The HIFiRE-1 simulations for SPARC use a series of meshes of varying resolution as inputs to pre-defined scaling studies. Our goal is to leverage these studies and eventually partner with the SPARC team in their performance and scaling studies. The SPARC team is already studying the *file-based* and in situ workflows, and in transit will be a third workflow in that study once development and testing is complete.

In simulating the HIFiRE-1 data, SPARC generates meshes for both volume and surface data which contain fields to record temperature and pressure changes the body experiences during the simulation. For each timestep, the Catalyst tool renders visualizations of field data over both the surface and volume meshes. These include a set of images that show the meshes and field data as computed on the surface and also as slices through the volume. These images are a product of parallel rendering during which multiple Catalyst processes communicate and coordinate to create each image.

IOSS is the enabling technology that underlies each workflow demonstrated in this work. At the application level, IOSS provides developers with an interface that describes mesh data with a geometric analogy applicable to both structured and unstructured meshes. IOSS also is equipped

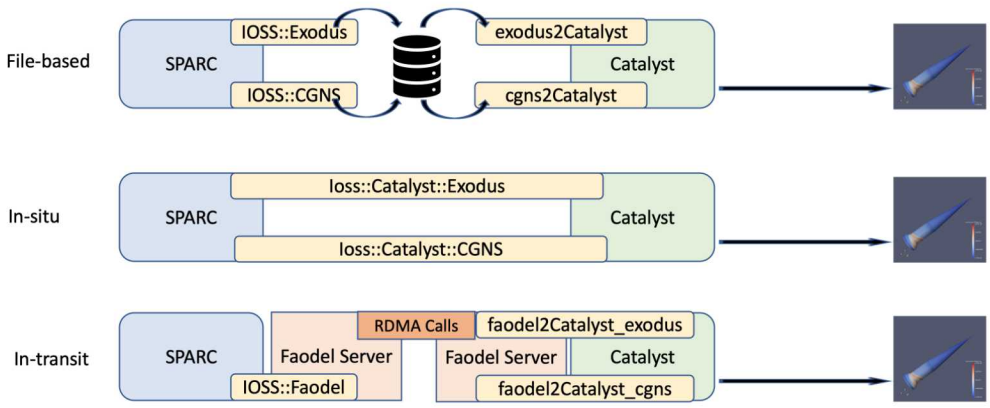

Figure 2-1. Software components of the *file-based*, in situ, and in transit workflows.

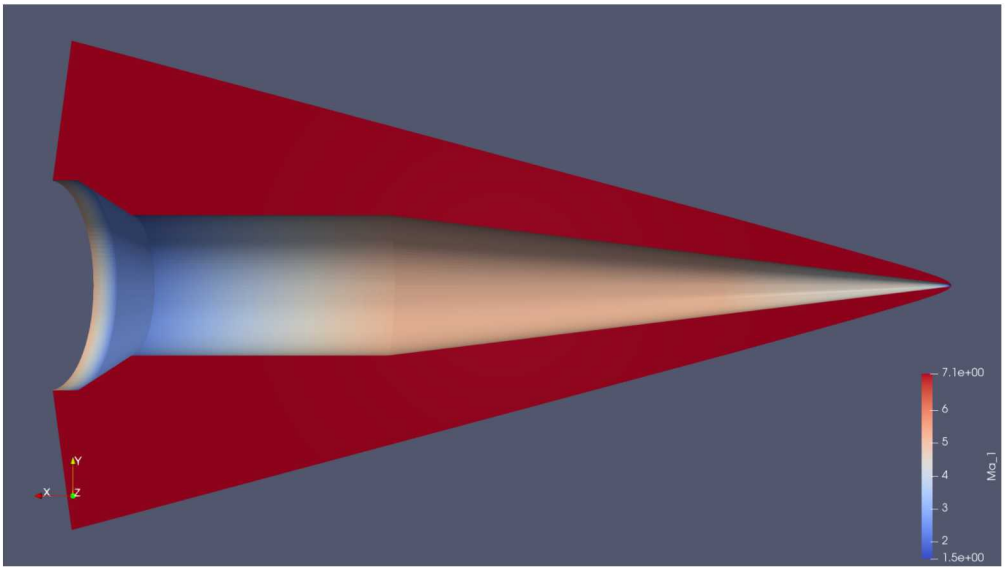

Figure 2-2. Volumetric Data created by faodel2Catalst

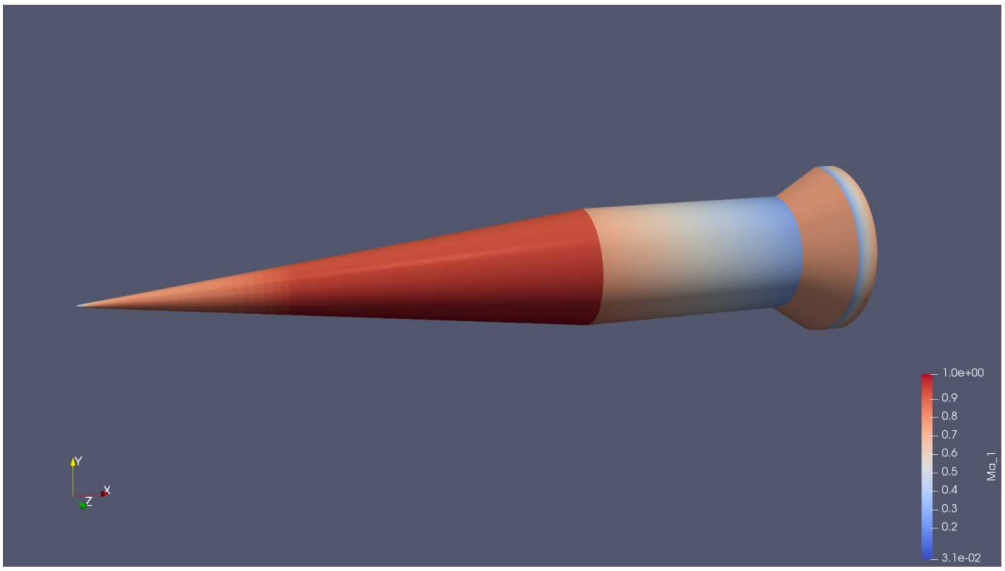

Figure 2-3. Surface Data created by Catalyst

with several "I/O backends" that communicate mesh data to and from specific mesh file formats like Exodus or CGNS. These backends are implemented in C++ as concrete descendants of the IOSS class `Ioss::DatabaseIO` and are specified by the application developer via the *factory* or *registry* software design pattern. This architecture makes it straightforward to change the mesh storage format that the application uses.

Although the IOSS interface is generic, the mesh storage formats are not necessarily interchangeable. Historically, Exodus has been used for unstructured data that is typically found in Finite Element Method computations and CGNS has been used for structured data that is typically found in CFD work on Finite-Difference and Finite-Volume techniques. Like most multi-physics codes, SPARC uses a hybrid mesh (containing both structured and unstructured mesh data), and so uses both the Exodus and CGNS storage formats to store mesh data. This is a consequence of the storage formats involved and not a limitation intrinsic to IOSS. There is an on-going effort to have Exodus and CGNS store hybrid meshes in their entirety and this has shown progress. In fact, SPARC has the option to write volume data to an Exodus file.

At the IOSS software design layer, the work described in this report includes the addition of three new database classes: `Iofaodel::DatabaseIO`, `Iovs_cgns::DatabaseIO`, and `Iovs::exodus_DatabaseIO` to the IOSS I/O factory mechanism. The first, `Iofaodel::DatabaseIO`, marshalls mesh data into the FAODEL data management layer. The second, `Iovs_cgns::DatabaseIO`, marshalls CGNS data into Catalyst for visualization. The third, `Iovs_exodus::DatabaseIO`, marshalls Exodus data into Catalyst for visualization. The last two have an additional qualifier for Exodus or CGNS; this follows the SPARC pattern where Exodus or CGNS names indicate surface and volume meshes, respectively.

Modifying SPARC to use these new IOSS interfaces to read and write mesh data enables each of the three workflows studied in this report. This approach also holds promise for how simulation applications themselves are developed. Clearly, the ability to read meshes produced by different meshing tools and utilities is an early step in the development process as is writing computed mesh data for later analysis. Using IOSS, a relatively stable interface, this I/O capability can fixed early in the development process and is available when later processes such as V&V are conducted. In this way, choosing the particular form of the mesh ouput or input will not affect how the simulation runs, only where the data ends up.

3. FAODEL

In order to accelerate the extraction of information from data, workflows that couple computational science applications are proving increasingly useful. By allowing developers to reason about their computational problems in a modular and regular manner, such workflows trade initial complexity in their definition and construction for ease of modification and increased reuse once they are in place. In order to maintain workflow construction as a viable technique for computational science, that complexity must be managed as projected system and data sizes increase.

Several potential complications are clouding the horizon, however:

- Impedance mismatches between data generation rates and parallel file system bandwidth have been an issue in computational science for some time now. System architectures have addressed this with various refinements to the memory/storage hierarchy, more recently involving the development of node-local "burst buffers". While these changes provide more options and flexibility for workflow designers, they also introduce added design considerations.

- Coupling simulation and analysis using in situ and ex situ techniques present a different set of choices for workflow designers. Depending on system loads, problem inputs, or changing availability of specialized compute engines, it may be advantageous to migrate analytics tasks in and out of a particular application. While this may not make a difference in the end state of a workflow, adding this capability presents significant data management issues.

- While the bulk-synchronous-program (BSP) model remains dominant, emerging decentralized decomposition and scheduling models are being explored for their potential to provide dramatically increased scalability. Assumptions underlying the design of workflows for a group of cooperating BSP applications will not necessarily hold for workflows designed around asynchronous many-task (AMT) execution. For example, using a parallel file system to exchange data between workflow components will likely be problematic as the number of discrete application tasks increases.

Our group has been exploring how changing the ways in which cooperating applications exchange data can provide leverage on these concerns, which are central to this milestone. This section describes a set of services for data management and exchange, FAODEL , for use in such applications.

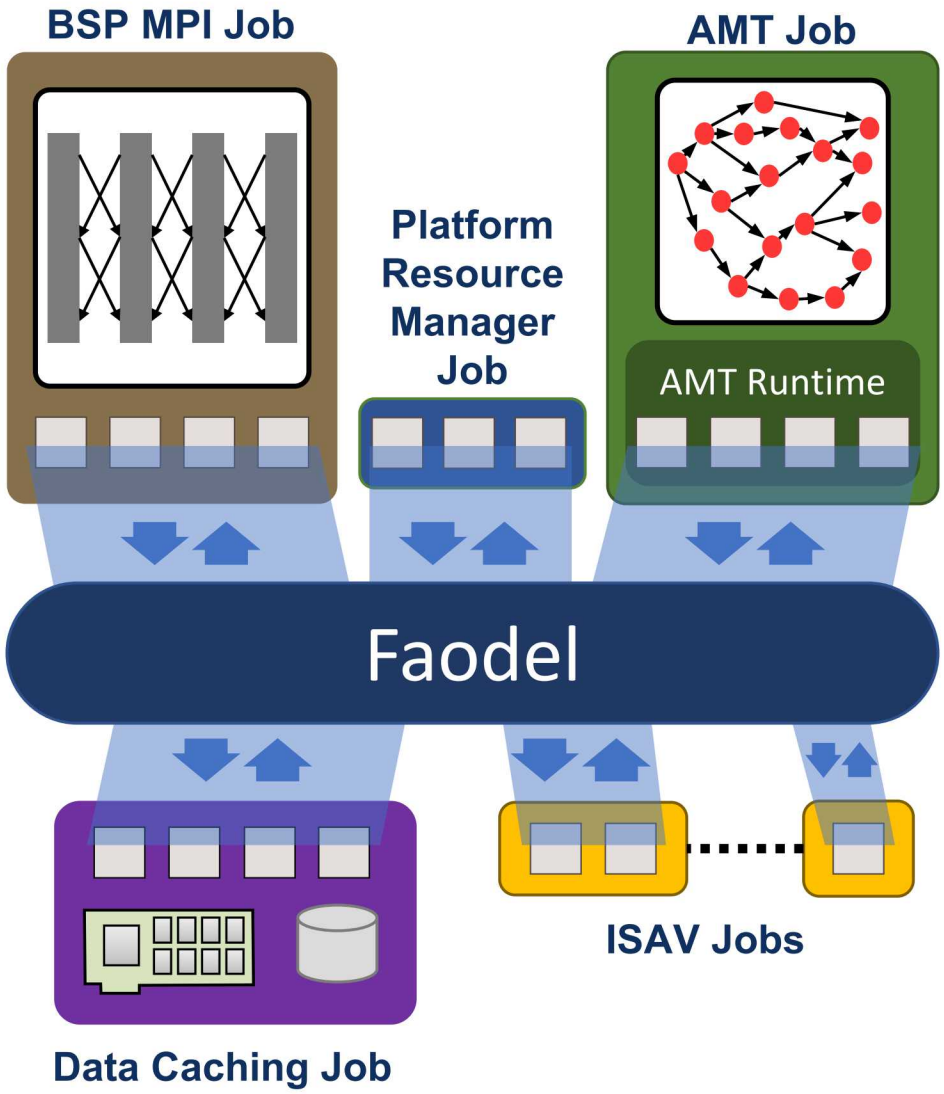

Figure 3-1. High-level diagram showing the relationship of FAODEL to a workflow comprised of an AMT application and in situ analysis and visualization. The large colored shapes encompass the resources used by each entity. The small squares represent computational resources (e.g., compute nodes, processors, or threads). The cylinders represent storage resources.

3.1. FAODEL Design & Architecture

The design of FAODEL is largely driven based on requirements derived from the operating environment of today's high-performance computing (HPC) platforms. As depicted in Figure 3-1, FAODEL provides a means of connecting several different jobs that run concurrently on a platform. First, BSP or AMT parallel simulation jobs run and produce data objects that are either stored in internal resources or published to other resources. While coupling and workflow scenarios may use FAODEL to pass data between components, this more complex use case has not been fully explored yet. Second, FAODEL may use distributed memory and NVM to absorb bursts of data from the application or replay results from simulations to requesters. Third, *in situ* analysis & visualization (ISAV) tools use FAODEL to retrieve and analyze data. Finally, coordination applications provide a means of helping the different jobs in the environment locate and connect with the resources of other jobs.

An examination of the application environment motivated three fundamental requirements for the design of FAODEL :

- FAODEL must provide basic primitives for users to reason about and decompose their datasets, but at the same time, the API must be as agnostic as possible about how developers manage their data. Rather than force users to design algorithms around a data store's indexing and migration policies, it is better to provide mechanisms for users to express how the system should manage their data. By offering mechanisms to control data epoch visibility and a simple key/blob interface, FAODEL offers an approach that can serve many kinds of clients.
- To aid scalability, separating application fates (i.e., the simulation from the analytics) also offers independent scalability through loose coupling. This requires using a communication layer that offers efficient data transfers between jobs while at the same time not breaking the communication libraries used within jobs (e.g., MPI). As such, FAODEL cannot simply rely on sockets or splitting an MPI communicator and must instead use a low-level Remote DMA (RDMA) communication layer. This layer is an evolution of the long proven NNTI layer from the Nessie RPC library [16, 19].
- FAODEL must provide a way of migrating data objects from memory to higher-capacity resources, such as burst buffers or the parallel file system (PFS). This requirement implies FAODEL must transition in-memory objects to systems with vendor-proprietary or file-based APIs.

FAODEL is made up of several software components. We describe the higher-level components most relevant to application developers in the remainder of this section.

3.2. Kelpie

Kelpie uses a distributed hash table (DHT) to provide a *key/blob* abstraction that facilitates flexible data exchange between different executables (e.g., simulation application and applications for visualization and analysis). A *key* is a programmer-defined text string that allows

the programmer to attach semantic significance to the associated data, a *blob*. Although a key attaches programmer-cognizable meaning (and possibly structure) to a blob, Kelpie is entirely ignorant of any meaning attached to a key or its associated blob. An example key might encode the application name, run number, iteration number, variable name, and some information about what part of that globally distributed array this blob represents. Separate processes can exchange data via Kelpie by exchanging key information. Key exchange can be explicit or implicit (i.e., keys can be constructed in a well-known way).

3.3. OpBox

OpBox is a library for implementing asynchronous communication between multiple entities in a distributed application. Our experiences with remote procedure call (RPC) libraries found that while RPCs provide a simple way to coordinate data transfers and invoke action at remote nodes, it is often difficult to coordinate more sophisticated data management services (e.g., ones involving more than two nodes, time-out conditions, or race conditions). Rather than leave the task of coordinating transfers entirely to the next layer up, OpBox provides the user with primitives for expressing a protocol as a state machine that the communication layer can process in an asynchronous manner. A communication pattern between a collection of nodes in OpBox is an `Op`. Users define and instantiate various `Op` classes as desired. Each provides a handle to the `Op` (implemented in C++ via `future`/`promise`), and a method to instruct OpBox to start running its encapsulated state machine. As each node in the `Op` communication pattern receives the `Op`, it processes the state machine accordingly.

OpBox provides a collection of operations that are common to many applications including ping and counter operations. In addition, OpBox includes a Directory Manager Service that can be easily incorporated into an application. The Directory Manager stores node information in a hierarchical directory. A typical FAODEL application would have at least one Directory Manager instance acting as a naming service to locate components of an application.

3.4. Lunasa

Lunasa provides user-level memory management services for network memory. For performance reasons, FAODEL relies heavily on RDMA to transfer data throughout an HPC system. RDMA eliminates the need to copy user data objects to kernel buffers and allows data transfers to occur without CPU intervention. RDMA transfers require the user register the memory buffers that are the source or destination of the transfer with the underlying network transport. Additionally, the virtual memory space that contains the memory buffer must be locked (or pinned) by the kernel to prevent it from being relocated in physical memory. The costs of registering and de-registering memory vary by network transport.

Lunasa amortizes these costs by facilitating the *explicit* reuse of registered memory. Existing approaches to reuse registered memory [24, 12, 4], do so *implicitly*. In other words, instead of de-registering memory buffers at the end of an RDMA operation, the registered memory is left

registered and its registration information is cached. Subsequent requests to re-register the cached memory can be fulfilled without incurring the costs of registration.[1] As a result, application programmers are encouraged to manage their memory buffers to extract as much reuse as possible. Lunasa frees application programmers from the burden of managing these implicit reuse semantics and allows them to *explicitly* request memory allocations for use in RDMA operations [10]. To accomplish this objective, Lunasa requests blocks of memory from the system and registers them at the outset. Lunasa then manages the resulting pool of registered memory as a standalone resource for satisfying user requests.

3.5. NNTI

The Nessie Network Transport Interface (NNTI) provides a portable, lightweight abstraction for RDMA operations on common HPC systems. The NNTI library was originally developed as part of Sandia's Network Scalable Service Interface (Nessie) RPC project to enable portability across HPC interconnects [16, 19]. In the following years, NNTI was spun off into a separate project so it could be used beyond RPC applications.

NNTI is built around four core concepts: memory buffers, send operations, one-sided operations and events.

Many HPC interconnects require memory regions to be registered with the NIC before the memory can be used in the data transfers—especially one-sided operations. In order to do DMA, the NIC must know the physical address of the memory region involved. When the application registers the memory region, the pages are pinned to prevent the Virtual Memory Manager (VMM) from relocating the pages and changing the physical addresses. NNTI tracks these memory regions and provides the application with a handle that can be shared with peers to perform data operations.

The NNTI send protocol is a messaging protocol used to transfer data from sender to receiver. The protocol uses command packets to initiate the transfer and tell the receiver the parameters of the message including destination, length, and event flags. The exact format of the command packet is transport specific, but it is expected that it contains enough information for the receiver to make decisions about message delivery.

The NNTI one-sided API is a lightweight RDMA API that is mapped as closely as possible to interconnects with native one-sided operations. On interconnects that do not have native one-sided operations, NNTI uses a protocol similar to the send protocol to manage the transfer. In addition to RDMA, one-sided atomic operations are provided.

All NNTI data transfer operations are asynchronous. NNTI events are generated at the completion of data operations and contain the detailed results of the operation. Completion does not mean success, so the event's result field must be check for each operation. Events are delivered to either an event queue or a callback.

[1] It is also worth noting that caching the registered memory eliminates the de-registration costs that would otherwise be incurred at the end of an RDMA operation.

3.6. Inter- and Intra-Job Communication

From a data services perspective, a fundamental challenge for exascale computing is that the community lacks modern tools for developing the services that are needed. Communication within jobs (intra-job) is largely a solved problem: traditional applications use MPI while AMT codes use low-level communication libraries such as GASNet. When joining multiple applications together, the standard practice is to merge the codes into a single job that leverages the same intra-job communication library. For MPI jobs this often involves using an MPI communicator splitter to create a partition of ranks for each application. The practical challenge of this approach is that the codes' libraries must be able to coexist at build time and/or run time. Similarly, a crash in one application can lead to a crash in all applications.

An alternate approach is to run the applications in separate job allocations and employ job-to-job (or inter-job) communication. This approach is appealing because it provides native isolation between applications and is amenable to workflows. However, inter-job communication is not prevalent in HPC for a number of reasons. First, there is a lack of communication libraries that make it easy to establish high-performance communication between jobs. Second, large-scale platforms such as the Cray architectures implement access control mechanisms for security reasons that make it difficult for users to perform inter-job communication. While necessary these mechanisms impede users with platform-specific certificate challenges.

3.7. FAODEL and the Storage Hierarchy

FAODEL provides applications with services for transferring data to storage resources throughout the system's storage hierarchy. Each tier in the storage hierarchy provides different access characteristics which provide benefits in different use cases.

Distributed memory provides access to the collective DRAM (conventional DRAM devices and 3D-stacked DRAM devices) within the application's hardware allocation. Relative to other storage resources, distributed memory provides low-latency, high-bandwidth storage. RDMA transfers allow for efficient access to remote memory resources. Distributed memory can be used by AMT runtimes, for example, to store and exchange application variables and by coupled codes to exchange simulation data.

Local persistent storage resources include SSDs and NVRAM. Locality varies by system. In some cases, persistent storage may be available on each compute node, other systems may provide per-chassis or per-rack persistent storage resources. Local persistent resources can be leveraged as part of a checkpoint/restart solution (*cf.* [18]). Similarly, because these devices typically provide much more storage capacity than volatile memory (i.e., DRAM), they may also be used in support of in situ analytics.

In most systems, the principal archival storage resources are provided by a parallel file system. Archival storage provides high-latency, low-bandwidth access to high-capacity storage devices

(e.g., hard disks). These resources are commonly an integral part of checkpoint/restart solutions.[2] They may also be useful for storing analysis output generated by in situ analytics tasks.

[2] Although checkpoints stored in local persistent storage can be used to recover from many failures, some failures may incapacitate a compute node such that its local storage is inaccessible. In these cases, checkpoints stored on global persistent storage resources can be used for recovery

4. CATALYST

Catalyst is an *in situ* library built using ParaView to link visualization capability directly with a simulation to produce visualization and analysis products as part of the simulation run. Catalyst pairs with ParaView by taking pipelines which are generated interactively within ParaView, including all views and algorithms, and exporting these into a pipeline which can be run as part of the simulation to generate those same views and algorithms. Many improvements have been made since Catalyst's first release into ParaView version 3 to both the usability of the pipeline creation and efficiency and scalability of library itself. While originally released as an optional part of ParaView's build process, Catalyst has since evolved into a separate library optimized for *in situ* operation, to be available with ParaView version 4.

4.1. SPARC Catalyst Integration State Before Milestone

The initial Catalyst integration with SPARC consisted of two output database types defined in the SPARC input deck. The first type, called *catalyst*, invokes an IOSS database for Exodus unstructured data to output SPARC surface meshes. This Exodus IOSS database implementation converts data from IOSS unstructured to Catalyst (VTK) and is loaded into the SPARC executable at run-time by means of a dynamic library that contains the Catalyst (VTK) specific code. The second type, called *catalyst-direct*, outputs structured mesh data (CGNS) to Catalyst and directly converts the data to Catalyst (VTK) data structures within the SPARC code. The direct style of output for structured data was used because IOSS handling of structured mesh data in parallel was still under development at the start of the SPARC Catalyst integration project.

Control of Catalyst output from SPARC is managed by a Python file referenced in the SPARC input deck. This Python control file can be exported from the ParaView GUI for a particular SPARC input mesh, or it can be a Sandia developed Phactori Catalyst control interface file. The Phactori interface allows users to define the operations and image output from Catalyst in terms chosen and default parameters for common visualization tasks.

During the first half of FY20, a Python based build system called STAMPS (Sandia Targeted Automated Make ParaView System) was developed to manage build and deployment of Catalyst with SPARC. The STAMPS system builds ParaView/Catalyst for linking SPARC with catalyst-direct and the run-time dynamic library necessary for IOSS Exodus Catalyst output. STAMPS can build ParaView/Catalyst on all SPARC supported platforms (CTS-1, CEE, ATS-1, ATS-2, TLCC2, Stria/Astra).

Catalyst support for the milestone required development four new capabilities to support post-processing, *in situ*, and *in transit* workflows utilizing SPARC, IOSS, and FAODEL.

1. ParaView/Catalyst post-processing of CGNS and Exodus data files written by SPARC
2. Structured mesh data (CGNS) output through an IOSS database to Catalyst to replace SPARC catalyst-direct
3. Programs that read CGNS and Exodus data through IOSS and output to Catalyst through IOSS using the database in requirement 2 to support FAODEL *in transit* workflows
4. Instrument SPARC to use the database in requirement 2 for *in situ* output of structured mesh data

4.2. Python Based Catalyst Post-Processing for SAW Workflow

To address the ParaView post-processing workflow for SPARC-generated Exodus and CGNS files, two Python programs were developed that utilize the ParaView `pvbatch` program. These programs, called `cgns2cat` and `exo2cat`, read in parallel CGNS and Exodus files, respectively, and output to Catalyst. The ParaView `pvbatch` program can be run in symmetric mode, which is identical to Catalyst running *in situ* while coupled to a simulation like SPARC. This allows these programs to accept input from a ParaView generated Catalyst control script or a Phactori interface control script. The Exodus and CGNS parallel data file read uses ParaView's readers to convert the data to Catalyst (VTK) format and does not use IOSS. A SAW workflow was developed that utilizes these programs to read SPARC Exodus output files and generate a default set of external mesh views using Phactori.

4.3. Catalyst IOSS Database for Structured Mesh Data

A new IOSS Database type called *catalyst-cgns* was developed to handle structured mesh Catalyst output through IOSS. The implementation of catalyst-cgns is similar to the catalyst-direct output type in SPARC. Additionally, a software refactoring was performed on the existing Exodus catalyst IOSS database and was renamed catalyst-exodus to clearly distinguish the two output paths for unstructured and structured data. The Catalyst pipeline control can now handle single and multiple pipeline inputs with both catalyst-exodus and catalyst-cgns mesh types. IOSS property options were added to control file writing to VTK vtm format of the Catalyst (VTK) mesh representations, which is useful for both debugging and generating Catalyst control scripts from ParaView. Both IOSS database types share a common dynamic library implementation that is loaded at run-time by the application code that uses the databases.

4.4. Catalyst IOSS Post-Processing Programs

Alongside the creation of new IOSS databases for Catalyst, two programs were developed to read Exodus and CGNS data in parallel through IOSS and output data to Catalyst using the new IOSS databases: `cgns2catalyst` and `exo2catalyst`. The programs have command line options to support various Catalyst control options and output the Catalyst (VTK) mesh representation to

VTK vtm format output files. The FAODEL IOSS database implementation for reading data *in transit* from a FAODEL data store can be easily coupled to modified versions of these applications by changing the IOSS input properties and database name for the read portion of the application.

4.5. SPARC Catalyst Integration Using New IOSS Databases

Once there were catalyst_cgns and catalyst_exodus database types in IOSS, it was necessary to alter the SPARC simulation code and input deck handling to correctly access these IOSS database types and manage all the concomitant controls and parameters. Essentially, this involved:

1. Adding various parameter options and parameters to the SPARC input deck schema in the volume-post-processing and surface-post-processing sections

2. Obtaining that information from the parsing and passing it through the SPARC execution flow to the appropriate usage points

3. Adding to the SPARC mesh writing code to handle the additional IOSS database types

4. Adding extra code to the SPARC mesh writing functions to deal with various specializations in the IOSS database types, e.g. IOSS properties to specify Catalyst scripts and usages of mesh_model_coordinates vs mesh_model_coordinates_x/_y/_z.

The catalyst_cgns database type was made to work first in the volume-post-processing section of the SPARC input deck. This feature was made to work first on CTS-1 systems (Eclipse), and then moved to ATS-1 systems (Mutrino). The catalyst_exodus database type was then made to work in the surface-post-processing section of the SPARC input deck.

4.6. Issues with Multiblock Parallelism in SPARC and Catalyst

It has become apparent than most research projects involving parallel I/O use single block datasets. That is, the meshes are a single structured or unstructured grid of data. While this simplifies the research by avoiding many issues not strictly bearing on the research in question, it also hides a plethora of real-world problems which appear in parallel in production problems which almost invariably involve multiblock data sets. A great deal of time and energy was spent in successfully working on parallel multiblock issues for Catalyst during this effort, and most of the work here transferred seamlessly to the FAODEL effort. The result is that the work done for this milestone will be much more quickly applicable to real-world problems than had efforts been confined to single block meshes.

4.7. Build Issues on ATS-1 (Trinity-like) Systems

Although it is mainly a technical detail, we should note that getting all the Catalyst-related (and FAODEL-related) pieces of this project built on the ATS-1 system presented many issues which had to be worked through. There were configuration issues, build issues, link issues, and runtime issues. The work presented in this report required the integration of several large software packages, including:

- ParaView/Catalyst
- SPARC
- Trilinos
- Seacas
- Ioss (in Seacas)
- FAODEL
- Ioss driver programs (cgns2catalyst, exodus2catalyst, etc.)

Each of these complex software package had to be configured, built and, in many cases, linked together. Significant technical work was required to integrate all of these packages to create a single, usable, high-level workflow.

5. WORKFLOWS

In simplest terms a *workflow* for scientific computing is an any-type data coupling between a data producer and data consumer. In most workflow researchers' eyes, workflows are distinguished from code-coupling by the ways in which the following characteristics are addressed:

- **Multi-machine deployment**. In the HPC sense, this means multiple platforms. A workflow would run on more than one capacity cluster, or ideally on a capability cluster (such as Trinity) and capacity clusters together. Policy challenges are frequently more significant than technical ones. For example, to automate logging into Trinity from another platform, permissions must be able to be scripted (eliminating the human in the middle). The capabilities an automated connection offers are likely strictly limited to avoid potential security errors. A fully capable workflow system would offer this as an option rather than requiring user intervention. This would enable using a capability resource for the largest compute tasks and then special purpose analysis clusters (for example, clusters with much higher memory-to-core ratios but far fewer nodes) for other processing tasks. This not only expands the types of codes which can be run, but also takes advantage of the right tool for each job. Which tool a particular task is assigned to can be determined at deployment time according to a particular run's parameters.

  With the size of pre-exascale and exascale machines, the expectation is that instead of running a single, large task, collections of tasks will run which offer workflow-like capabilities. These kinds of deployments, if loosely coupled, work and act like a traditional workflow. The work demonstrated as part of this milestone follows this model, showing three different ways to achieve this integration. The *in situ* approach is not loosely coupled, but it seeks to achieve the same ends, minimizing data movement while accepting other tradeoffs.

- **Orchestration**. Different workflow engines handle this differently, but there should be some ability to control what components are running at the same time, if any. For example, with Kepler, a serial or parallel orchestrator controls what runs when. By switching out the orchestrator component on the workflow, the way the workflow runs can be changed. The ability to have *fan-out* and *fan-in* operations is crucial. Gatekeeper monitoring functions are required that collect input values to make decisions or routes outputs to a collection of processing components. The key idea is that where possible, having components execute simultaneously offers faster time to processing than a purely serial approach. The demonstrations for this milestone show different ways orchestration can be achieved, including tightly via *in situ* processing and loosely via FAODEL.

- **Monitoring**. Seeing the status of a workflow in process is important to understand what is happening. Some HPC workflow engines have this capability, with varying degrees of

granularity and accessibility. A visual representation of the workflow makes clear what components are running and can potentially also allow inspection output codes and other status indicators. For debugging or understanding a workflow, this is crucial. The Dask toolkit [20] offers a version of this for Python-based tools. FAODEL offers these tools via the Whookie web-browser based monitoring interface.

- **Portability to other underlying infrastructure**. While there has to be some base compatibility for code to run elsewhere, it is desirable to minimize changes to the workflow or the code to move to a new platform. For example, moving from Slurm to PBS as a job queue manager should not require any changes. Instead, the appropriate component may need to be swapped out by the user, or ideally for a good system, the workflow system knows and swaps things out automatically without user knowledge or intervention. There are limits to this, of course, but it is something all workflow engines strive to offer.

- **Flexibility to trivially handle small changes**. Workflows should be parameterized so that necessary metadata, such as the input deck for a run, can be set externally and inserted where appropriate. Workflow components should be able to accept these parameters, within reason, and operate without changes. Having to adjust the workflow for each small change restricts the workflow to a single configuration. Instead, a workflow should offer a generic framework for performing a series (or parallel) set of processing steps to achieve an outcome. It should not include as that the particular pieces that describe a particular run. IOSS and FAODEL offer these capabilities by abstracting away details and relying on configuration files and directory services to fill in specifics at runtime.

- **Data management capabilities**. Workflows, at their heart, are really more data management engines rather than processing management engines. Task-based programming is similar, involving the management of data creation, availability, and placement to schedule tasks. The data might be the return code fed into a conditional to decide which component to run next, but it is a data-based process management approach. In most cases, the goal is some operation that causes a sequence of operations to either transform data or execute the proper set of processing based on the state of components as the work progresses. Large scale data storage is not really what a workflow does well, one way in which HPC workflows are different from general workflows.

  Further complicating this picture for HPC workflows are the data sizes. Instead of items in the KB to MB range, they can easily be in the TB to PB range. Managing the movement of consistently large, distributed data items from one place to another requires special effort, an issue which has been addressed by researchers with the Doubly Distributed Transactions [14, 13] system. D^2T manages parallel operations, such as MxN data transfers ensuring that everything is both complete and correct.

- **Component-to-Component communication and signaling**. While it is possible to simply use the parallel file system to do all communication and signaling, that is just one option. Below we present more comprehensive options with some of the tradeoffs of each.

  - **Use the parallel file system**. Most HPC workflows today are written by application scientists unfamiliar with the extensive work on decoupled messaging and data

staging/transfer capabilities of modern platform. Further, it is simpler to read and write files as the independent processes do rather than integrate more advanced functionality. This results in files being used both to stage data between components as well as special files being created to signal to the next component that a data set is ready or what kind of processing to perform on the next data set. On the receiver side, constant file system contents monitoring is required to notice the changes imposing performance penalties on the whole storage array affecting all users. This is far from an ideal solution and one that this milestone demonstrates is strictly not necessary.

- **Abstracted I/O API**. By using an I/O abstraction, such as ADIOS [15] or IOSS, instead of directly using POSIX or HDF5 calls, the particular implementation used to handle the data movement tasks can be replaced without affecting the surrounding code. Instead, a simple re-link of the new library is sufficient to incorporate a new data transfer technique. In this milestone, IOSS is demonstrated as it is the main interface Sandia codes use to read and write mesh data.

- **Special purpose APIs**. In some cases, using particular APIs for signalling between components or between a component and the workflow system can offer many advantages. For example, the component can inform the workflow engine directly what the output state is so that the proper next component can be invoked. Without such a system, this information has to be inferred. Since low impact changes to the applications and analysis components is the goal of this milestone, this approach was not attempted.

While the above list offers a broad set of features, it is not comprehensive. For example, data management activities for files and other persistent objects require special care. Most systems cannot properly handle these activities and make a best effort. Underlying changes to data location, names, or other features that happen outside the scope of the workflow system are typically not able to be captured, leading to a potentially fragile system. No system today, unless fully integrated using a tool like a database for all data storage, can realistically handle this more advanced requirement.

Newer technology, like containers, offer promising ways to package and orchestrate workflow components. Kubernetes orchestrates container deployment at large scales, relying on each container to be self-contained and so eliminating concern about any particular local software state.

Resilience issues for workflows are still a research area with only brute force solutions being more widely available. Properly annotating a workflow in order to enable smarter resilience operations is not well understood and requires considerable additional work for a robust solution outside of a single domain niche.

The largest challenge with workflow systems today is that each scientific domain is creating a custom solution tailored to their own needs eliminating the complexities of general solutions. Developing general workflow solutions that are as easy to use as a domain specific tool is a daunting task that all but eliminates any adoption outside of the workflow system development partners.

The FAODEL family offers additional tools and technology to create a loosely-coupled workflow. This milestone explores some of these use cases to enhance Sandia production workflow needs. It is one of the ways to achieve the component communications described above.

Given this workflow capabilities list, the approaches used for this milestone are discussed in more detail in the following sections.

5.1. *File-based*

Currently, workflow are largely developed by hand and for specific purposes. This approach is far from ideal given contemporary and future data sizes and data creation rates; storage arrays have neither sufficiently large capacity nor have sufficiently large bandwidth to handle the loads at a frequency that best suits the science and engineering tasks being explored. This is the base case that this milestone demonstrates is no longer necessary with minimal changes to existing components.

5.2. In situ

The In Situ Terminology Project Report [3] details the various ways in which two different components can interact in situ. For this milestone, in situ refers to a direct memory-to-memory transfer from the simulation to the analysis engine. While other approaches are possible under the broad definition of *in situ*, for our HPC application purposes, memory pressures frequently require this model.

5.3. *In transit*

The *in transit* case incurs an additional data transfer cost since there is a staging area in the middle, but it enables decoupling the simulation from the analysis offering independent failure domains. Further, it can eliminate any processing speed mismatch concerns that could stall the simulation or overload the memory on the analysis nodes. It is not without potential faults, but most of these can be managed. For example, memory pressures in the staging area can be overcome by spilling excess data to some larger capacity storage such as additional compute nodes, local burst buffers, or even centralized parallel file storage. Additionally, resilience is more easily incorporated into this model since the staging area has knowledge of what was successfully handed off from the simulation to staging and what staging successfully processed. FAODEL offers these kinds of features as demonstrated in the milestone.

5.4. Workflows foster algorithm development

One side-effect of standardized, general workflows is that application scientists can focus on one part of their process at a time and mix and match different algorithm options with fewer concerns about how this will affect other components. Through decoupling or loosely coupling components behind standard interfaces, different options can be tested quickly with few fears of immediate failure. Instead, specific parts of the workflow, now isolated, can each be optimized accelerating not just the scientific inquiry, but also improving tools for general use.

The *in transit* approach demonstrated for this milestone offers the decoupling advantages described above. Specifically, when analysis is coupled *in situ* with the application, they share memory, CPU, network, and most importantly, failure domains. Should either the analysis or application fail, both fail. The pressures on node memory can also force a thinner, more widespread data distribution to fit both the computation and the analysis on the same nodes. This forces more communication and shorter computation phases generating less efficient workflows. The *in transit* approach addresses these limitations.

First, by separating the application from the analysis, either can fail without causing the other to necessarily fail. While it is true that the workflow must be able to compensate for the failure and restart of any component for it to be truly resilient, unless the components are decoupled like the *in transit* workflow, there is not a viable option for separating the failure impacts.

Second, the memory and network pressures are relieved. The computation can maximize the memory footprint it uses maximizing the computation time between communication phases and reducing the number of nodes necessary to run a job. The *in transit* nodes can stage the data to the separate analysis routines. Since the data staging routines offered by FAODEL are relatively lightweight compared to the computation or analysis code, far more data can be stored per FAODEL node than can be held for computation or analysis.

6. RESOURCE STUDIES

As a means of better understanding the tradeoffs of using the different workflow styles we constructed multiple experiments that focused on different aspects of data transfer within the workflows. All of these experiments focused on connecting a parallel SPARC simulation to analysis operations implemented in Catalyst. Given that using FAODEL is a new mechanism for implementing *in transit* workflows, we placed more emphasis in this exploration on characterizing how FAODEL performed in different scenarios. End-to-end performance characteristics for the three workflows is summarized at the end of this chapter.

In compliance with the U.S. Department of Energy's policy for digital research data management[1], experiment data collected in this work has been retained in the following repository:

`https://cee-gitlab.sandia.gov/dsva/fy20-l2-experiments`

6.1. Examining the Content of IOSS Data Exported through FAODEL

An important part of understanding the performance characteristics of the *in transit* workflow is inspecting how different datasets are decomposed and distributed by FAODEL. Library developers take into account a number of factors when determining how to split a dataset into labeled objects that FAODEL can use. For example, should multiple data structures be packed into a single object to maximize throughput or split into multiple objects to minimize unnecessary data in retrievals? How descriptive do object labels need to be in order to ensure data is discoverable? How should labels be picked to control how data is distributed across FAODEL pool resources?

The initial version of the FAODEL database for IOSS focused on a conservative implementation that splits data structures into their own objects. It also uses a verbose, descriptive labeling scheme to help developers get a firm handle on what objects IOSS/FAODEL produces when SPARC generates CGNS and Exodus data. This section explores the content generated in a normal SPARC run to help determine how future IOSS/FAODEL implementations can be improved.

[1] `https://www.energy.gov/datamanagement/doe-policy-digital-research-data-management`

6.1.1. Generating Simulation I/O Traces

For these experiments we enabled FAODEL's new tracing capability to get a timeline of when SPARC writes output to IOSS, which in turn generates and publishes FAODEL data objects that are transmitted to FAODEL pools in the system. The tracing capability works at the *client side* of a pool and records the time at which the client invokes the operation, as well as information such as the key name and object sizes when known. We configured SPARC to run on 32 nodes and export both volume and surface data in CGNS and Exodus formats respectively.

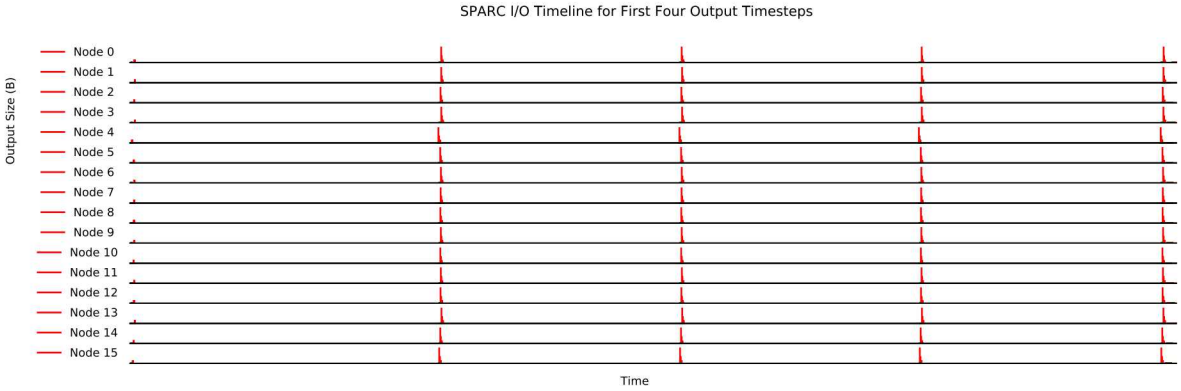

Figure 6-1. FAODEL's tracing capability recorded I/O activity for the first 16-nodes of a SPARC run. This trace includes IOSS database initialization operations during startup and four timestep operations. The amplitude represents the amount of data published in an individual operation at a point in time.

Figure 6-1 depicts the timeline of IOSS/FAODEL activities for the first 16 nodes of the SPARC run. After a small amount of activity at the beginning of the simulation for initialization, FAODEL only generates traffic at the four points in time when the simulation writes out timestep data. As expected, these writes take place at the same time due to the synchronous nature of the simulation. This synchronicity is the defining characteristic for HPC I/O, producing a massive amount of data that must be received and stowed away as quickly as possible in order for a synchronous simulation to proceed (performance implications of this behavior for *in situ* analytics have been studied in [11] and elsewhere). Burst buffers were introduced in HPC platforms to make traditional I/O fast enough to absorb this crashing wave. However, systems such as FAODEL offer an alternative where data can be trickled out as needed by downstream applications.

A more detailed view of the I/O patterns for the volume and surface datasets is presented in Figure 6-2. During startup (a), both datasets issue a series of publish operations to push necessary information such as dataset properties that consumers will need to access and interpret the dataset. A normal timestep operation (b) typically produces a collection of larger objects such as transient field data. An inspection of this data reveals that there is a definite amount of natural skew that takes place between ranks. While a portion of this skew can be attributed to the best-effort nature of FAODEL's tracing facilities, it is expected that simulation nodes will drift from each other until a synchronization point due to compute load or variations in hardware.

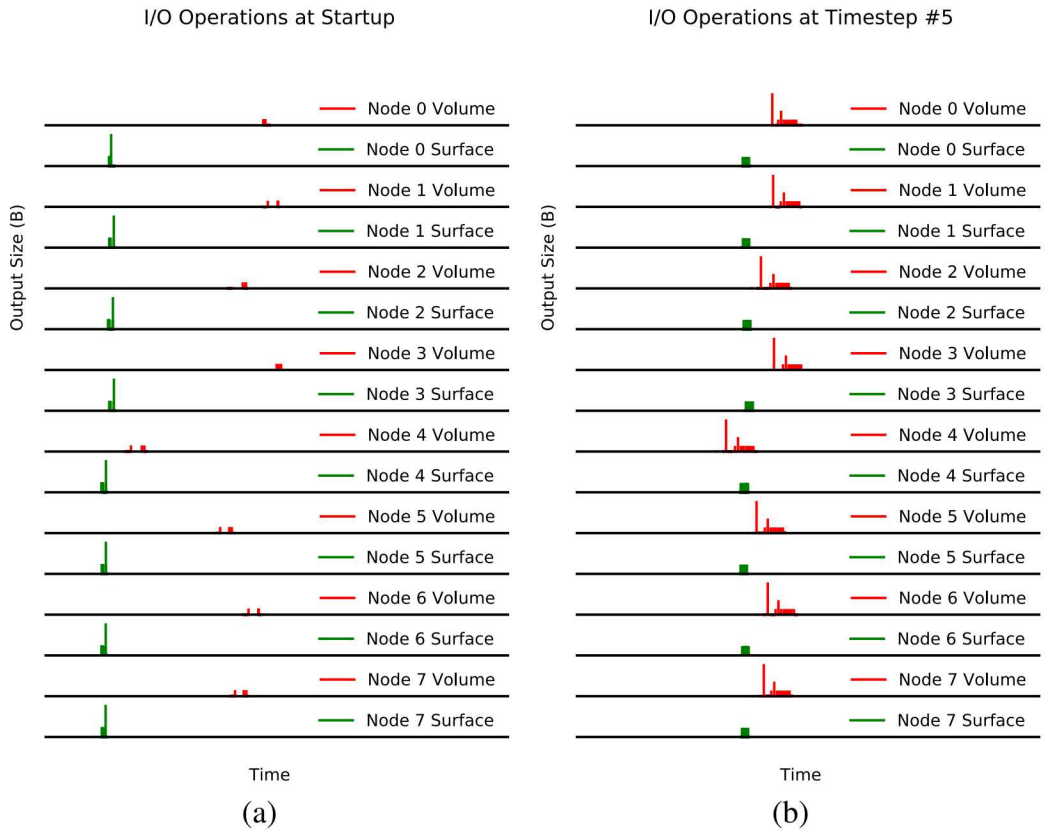

Figure 6-2. The surface and volume datasets produce different I/O patterns during (a) the startup portion of the simulation and (b) the normal output phase of a timestep. The amplitude reflects the size of an individual object being published. Volume and surface amplitudes are plotted separately due to significant size differences between the two.

6.1.2. Object Sizes

We analyzed the FAODEL traces generated by the SPARC simulation to get a better understanding of the content being generated during the simulation. Figure 6-3 provides a histogram for all the object sizes that were observed for the surface (a) and volume (b) datasets. Small objects (less than 1KB) dominate both datasets due to the number of properties that are written out as individual items. The volume data has objects which are roughly two orders of magnitude larger than the surface data due to the nature of the data.

6.1.3. Object Key Sizes

We also analyzed the FAODEL traces to get a better understanding of the size of the keys used to uniquely label each object with a practical workload. The current IOSS/FAODEL implementation uses a descriptive label to make the data more searchable and includes a variety of information about the object such as the block name and region type. Figure 6-4 provides the histograms of key length for the surface and volume datasets. In both cases we see that keys can be considerably

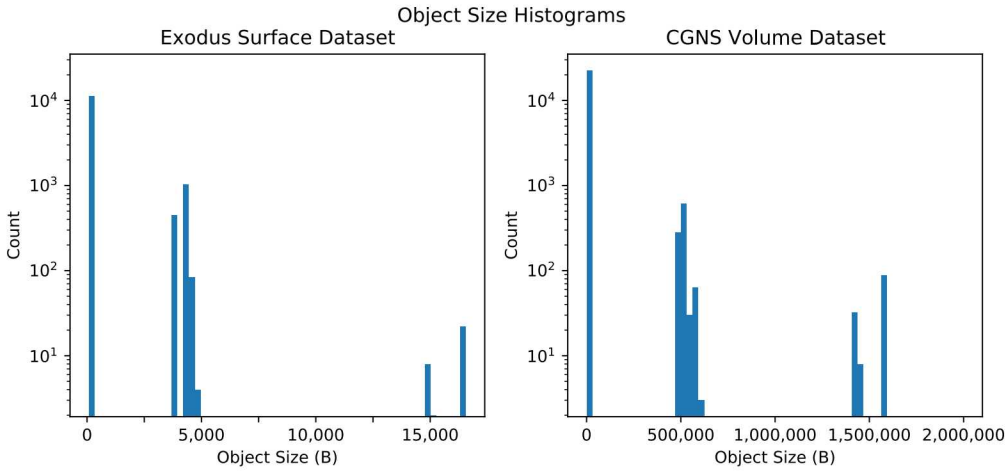

Figure 6-3. The histograms of object sizes found in the (a) surface and (b) volume datasets show that different object sizes may vary based on data requirements.

long (100 bytes). The keys for the volume dataset have more variety in lengths than do those for the surface data. This trait can be attributed to naming convention differences between the CGNS and Exodus libraries.

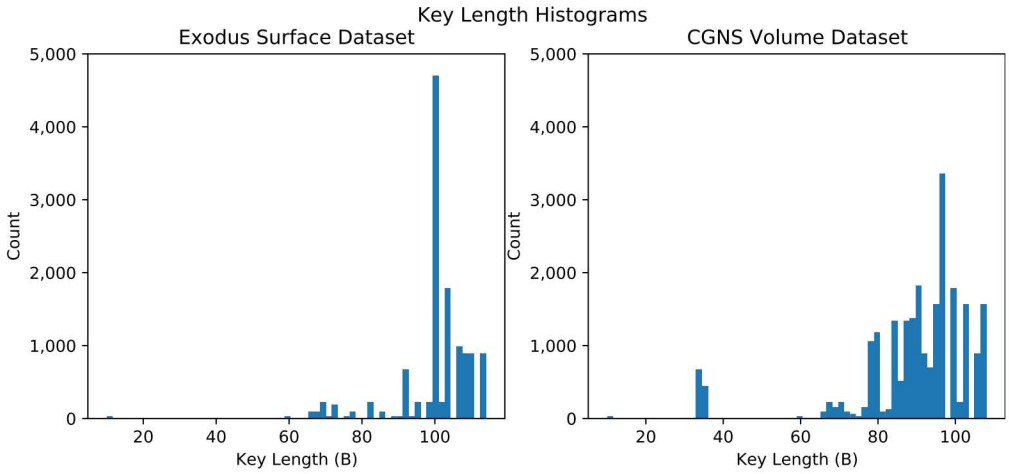

Figure 6-4. The histograms of the object key labels found in the (a) surface and (b) volume datasets show that the current IOSS/FAODEL implementation uses long keys for labeling objects.

6.1.4. Discussion

The intent of the initial version of IOSS/FAODEL was to build a general system for housing the different types of data that different IOSS applications need to store. Given that we did not know what quirks we would find in the data streams until we started handling real application data, we designed the encoding mechanism as conservatively as possible and oriented it to being capable of being searched and debugged by humans as needed. This choice means that our implementation may generate an excessive number of small objects. In many cases the *key length is larger than*

the actual data value. Both of these traits create a high workload for FAODEL. Using many small objects (1) increases the amount of bookkeeping FAODEL must do to index all items, (2) adds to client load by increasing the number of objects that must be tracked during transmission, (3) reduces the efficiency of network transfers, and (4) places strain on the actual file system that stores objects. However, it is important to observe that even with this stress inducing workload, IOSS/FAODEL was still able to move the data between systems and applications.

6.2. FAODEL Node Scaling

One of the advantages of FAODEL is that it provides workflow designers with an easy-to-use mechanism for controlling the amount of resources in the compute platform allocated for hosting intermediate data products. Users may want to scale resource pools in their allocation based on dataset sizes, throughput targets, persistence requirements, or host availability at runtime. It is therefore important to explore the impact of different scaling options on FAODEL's performance.

6.2.1. SPARC-to-FAODEL Experiment

We conducted a set of experiments to observe how the number of nodes and persistence requirements affected SPARC performance. Similar to the previous experiment, we connected a 32-node (1 process per node) SPARC job to a FAODEL pool and directed the simulation to write both surface and volume data through IOSS/FAODEL to an external pool. We varied the number of nodes used to host the FAODEL pool and adjusted whether the pool nodes wrote the data to a Lustre file system or not. We modified SPARC's performance counters to measure how much of the simulation's run time was spent in I/O operations to observe the impact of the different configurations on the simulation. While this approach does not capture the end-to-end timing of an *in transit* analysis pipeline, it does illuminate overheads that can impede the simulation.

The timing breakdown for the two experiments is depicted in Figure 6-5. The top set of experiments varied the FAODEL compute nodes from 1 to 16 and wrote incoming data objects to both RAM and disk. The penalty for writing to disk was significant and accounted for *over half* of the wall clock time of the simulation when the 32-simulation nodes wrote to a single FAODEL node. Adding more nodes to the pool significantly improved performance, until plateauing at 8 nodes. While 16 FAODEL nodes decreases the amount of time spent in I/O, the total time remains constant implying that there are other timing issues for the application that complicate scaling. As the middle plot in Figure 6-5 illustrates, switching to burst buffers significantly improves performance. However, FAODEL's I/O times still impose a noticeable penalty on SPARC performance. Finally, transitioning to a FAODEL pool that only stores data in memory yields the best performance. As seen in the lower plot of Figure 6-5, RAM-based pools do not impede SPARC performance. Performance for the 32-node simulation is maximized with a four-node, RAM-based FAODEL pool.

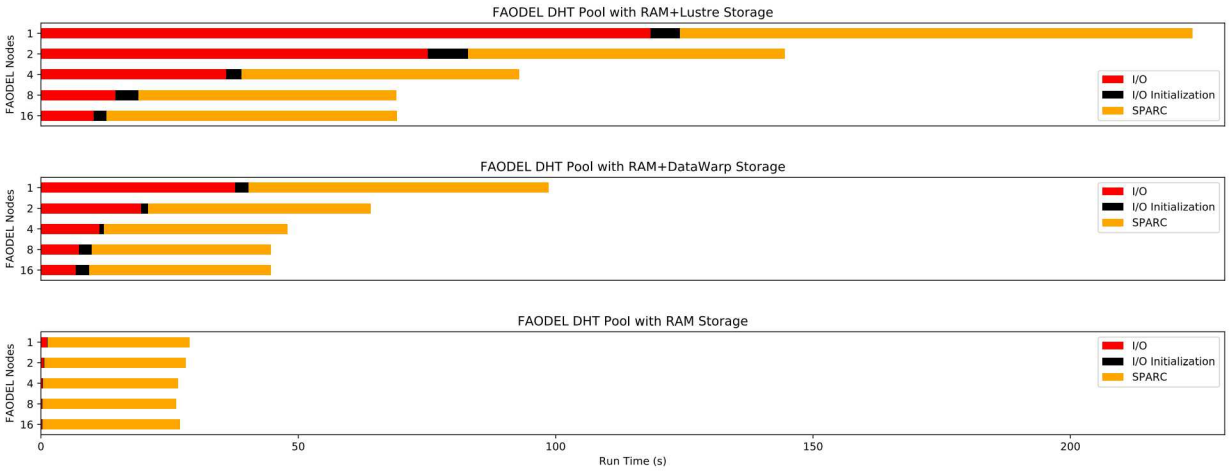

Figure 6-5. The performance for a 32-node SPARC job varies depending on the number of nodes in the external FAODEL pool and whether the pool writes to Lustre (top), the DataWarp burst buffer (middle), or plain memory (bottom).

6.2.2. FAODEL-to-Catalyst Experiment

The last step in the *in transit* process is to transfer data from a FAODEL pool to an application such as Catalyst for analysis. The set of faodel-to-catalyst programs were instrumented with timers to provide information about how long it takes a consumer application to use the IOSS library to retrieve data from a remote pool and pass it to a ParaView script for analysis. In this experiment we varied the number of FAODEL pool nodes used to host the data while the FAODEL-to-Catalyst program read and analyzed the data. We initially ran the FAODEL-to-Catalyst program with a single rank. As depicted in Figure 6-6, using multiple FAODEL nodes did not improve performance in this scenario because the single-rank consumer cannot take advantage of the parallel hardware due to the way data is distributed. Specifically, IOSS/FAODEL currently uses a coarse-grained data distribution strategy that places all data generated by a particular SPARC rank on a single server. As such, an analysis application that only retrieves one rank of data will only interact with one FAODEL server. This limitation will be remedied in future work by using a finer-grained distribution policy that distributes data more evenly.

The faodel-to-catalyst program was also run in 32 ranks to observe the impact of scaling the FAODEL nodes when using a parallel consumer. As depicted in Figure 6-7, we see that the parallel version runs faster than the serial implementation, and that adjusting the number of FAODEL servers can influence performance. In particular, the analysis of the CGNS volume dataset yielded its best performance when 4 FAODEL pool nodes were deployed.

6.2.3. Discussion

We investigated where the bottlenecks are in the system and found two issues in the current implementations of FAODEL and IOSS/FAODEL. First, the commit-to-disk stage of a remote I/O operation in FAODEL is in the critical path and must complete before acknowledgement is

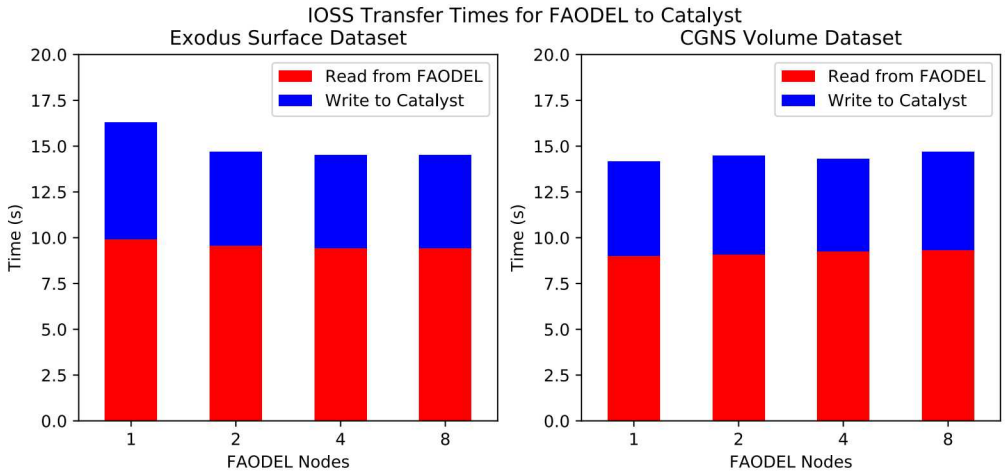

Figure 6-6. The faodel-to-catalyst application reads (a) Exodus and (b) CGNS datasets from FAODEL and pushes the data into Catalyst (1-rank)

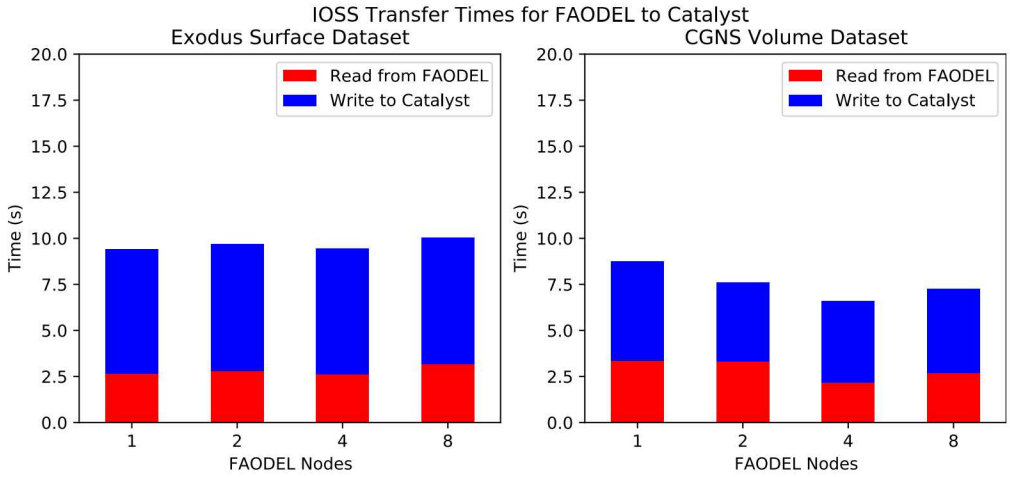

Figure 6-7. The faodel-to-catalyst application reads (a) Exodus and (b) CGNS datasets from FAODEL and pushes the data into Catalyst (32-ranks)

transmitted to the sender. While this design choice ensures the system will not prematurely report data is committed, it also impedes performance. In future work we will explore alternatives that allow a relaxed consistency model that allows commits to be deferred. Second, the current implementation of IOSS/FAODEL largely issues transactions in a serial manner. This approach limits FAODEL's ability to schedule concurrent transactions and reduces performance. We will investigate mechanisms to increase asynchronous communication in future work.

6.3. Build Factors

As software developers we spent a considerable amount of time this year modifying, configuring, building, and running the SPARC software stack. This task was challenging due to the complexity of the stack's libraries and the platform-specific issues of the build environment. Given that

developer time is a significant resource cost in the ASC budget, it is worthwhile to assess how software development factors influenced our productivity. While sorting the time spent on different software development activities into bins is not a straightforward task, *build time* is both amenable to measurement and provides insight into the overheads associated with this work.

We instrumented the build script discussed in Appendix A to determine how much time is required to properly build the FAODEL, Trilinos, and SPARC components used in our software stack. This script (1) retrieves each software library from its Sandia git repository, (2) configures each library's build system to use a specific combination of compilers and libraries that are available on the platform, (3) builds each library, and then (4) links all code into an executable that can be run on the platform. The current script builds shared libraries to enable software developers to rebuild a specific library without having to relink the entire application, provided the library's APIs remain unchanged. In previous years, platforms required static builds that necessitated a rebuild and relink of the top-level application any time a library changed.

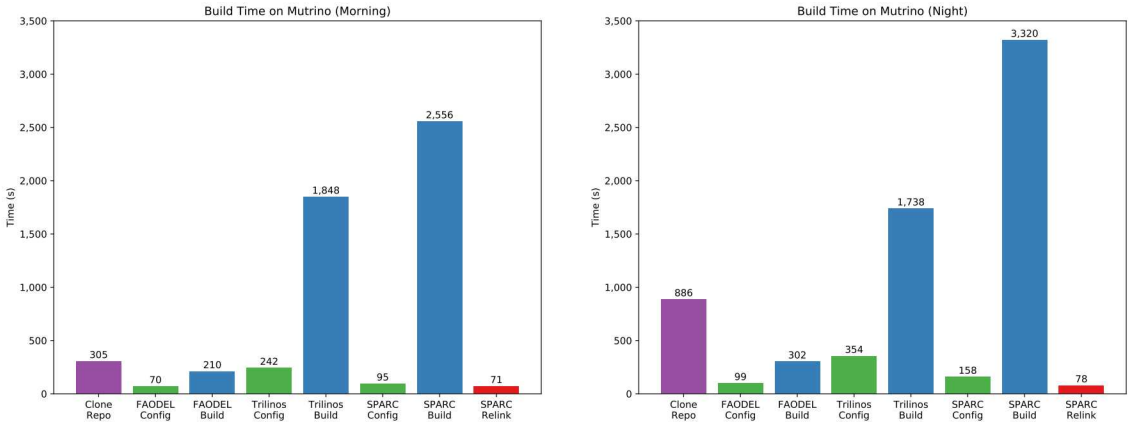

Figure 6-8. The amount of time to build components in the SPARC stack is significant and can vary based on whether the platform is (a) lightly or (b) heavily used by others.

Figure 6-8 provides timing information for a complete build on a Mutrino login node at times when the system is (a) lightly and (b) heavily used. The total amount of time required to build the stack is roughly 1.5 to 2.0 hours, with the dominant overheads being the Trilinos and SPARC builds. While we did not normally need to rebuild and relink the SPARC application due to our use of shared libraries, it is important to point out that small changes to any component in the stack for a static build should be expected to take at least a minute and a half to complete. This delay impedes the iterative development cycle and provides additional motivation for *in transit* approaches, in which separate executables (with separate build dependencies and iteration cycles) are used for the application, data management, and analysis components. This decoupling of the software involved allows for more independent design and development of the different components, as long as the data interfaces between them are well-defined.

6.4. Workflow Performance Summary

We have implemented all three workflows on the Mutrino platform at a small scale to demonstrate that our software components are mature enough to be used to route data between the parallel SPARC application and Catalyst's parallel rendering facilities. In this section we summarize different performance characteristics we observed while working with each workflow.

It is important for us to state up front that *the* in transit *work is still early* and that we have had to limit our experiments to avoid known problems. These limitations include the following:

- **Manual Launching:** Due to development timelines, the work reported in this section was conducted in a manual fashion where different workflow stages were launched by hand in different allocations. This process enabled us to adjust settings and monitor internal job information that is not required by normal workflow users.
- **Lack of *in transit* Overlap:** The current IOSS/FAODEL implementation does not publish timestep information until the end of the job. As such, consumers such as the `faodel2catalyst` job will not see the dataset until the job completes. This trait is a significant impediment to performance, but can be remedied by modifying the IOSS/FAODEL software to publish updates to the timestep value when a timestep closes. While it is expected that concurrent writing and reading will impact performance, we expect that it will not add significantly to the total workflow time.
- **Limited Experiment Sets:** Due to hardware availability and limited time, we currently do not have a broad enough set of experiments to provide the in-depth details we desire for our own interests as systems researchers. The reported measurements do not include performance data from Mutrino's Knight Landing (KNL) partition or a detailed assessment of burst buffer performance.

The primary application driver in this work is to have a 32-node SPARC job run the HIFiRE-1 problem for 20 timesteps and generate both volume and surface output. In each workflow Catalyst is then used to read the data and produce images about the simulation results. Table 6-1 provides a breakdown of how much time and memory is required to complete each stage in the different workflows. Timing values are wall clock times that include startup and completion costs for the individual stage. Memory estimates are based on the resident set size (RSS) the OS reported for a single rank at the end of the simulation. While there are known issues with RSS measurements on Mutrino due to the way huge pages are counted, the measurements are taken in the same manner and give a means of comparison between different tests.

Figure 6-9(a) provides a comparison of the end-to-end runtimes for the different workflow scenarios. A number of observations can be made from these initial performance experiments:

- **Burst buffers improve performance:** Mutrino's burst buffers provided a 20% improvement in SPARC's performance compared to Lustre. We observed multiple instances during our development where the Lustre file system was unusable due to activity by other users while the burst buffers remained performant.

Workflow	Stage	FAODEL Nodes	Time (s)	Maximum RSS (MB)
File-based (Lustre)	SPARC w/ IOSS/CGNS	n/a	18.542 s	180.611 MB
	cgns2catalyst	n/a	4.549 s	161.680 MB
File-based (DataWarp)	SPARC w/ IOSS/CGNS	n/a	14.555 s	189.132 MB
	cgns2catalyst	n/a	4.275 s	161.684 MB
In situ	SPARC w/ IOSS/Catalyst	n/a	42.451 s	301.962 MB
In transit	SPARC w/ IOSS/FAODEL	2	30.384 s	189.912 MB
		4	30.335 s	189.892 MB
		8	30.184 s	189.784 MB
	faodel2catalyst_cgns	2	7.625 s	367.001 MB
		4	6.612 s	367.001 MB
		8	7.260 s	367.332 MB
	faodel2catalyst_exodus	2	9.691 s	307.801 MB
		4	9.459 s	308.227 MB
		8	10.040 s	308.633 MB

Table 6-1. Timing information for different stages in the three workflow types.

- ***In situ* may adversely impact simulation performance:** While the *in situ* case is by far the most convenient method when analysis is desired (just launch and collect results), it took more than twice as long for the *in situ* simulations to complete than the cases where no analysis was needed in our experiments. However, it is important to note that this is an unfair comparison, as the above experiments were not run long enough to overcome startup costs.
- ***In transit* was better than anticipated:** As discussed in this report, the *in situ* implementation was largely targeted at delivering a new capability as opposed to achieving optimal performance. As expected, SPARC simulations using FAODEL were slower than simulations that simply wrote to disk. However, the FAODEL solution was only $2x$ slower than the native simulation, and faster than the *in situ* approach (Figure 6-9(a)).
- ***In situ* adds to memory overhead:** As illustrated in Figure 6-9(b), SPARC's memory use was roughly $1.5x$ larger in the *in situ* case than the file-based case for this particular visualization task.
- ***In transit* shifts memory and compute burden:** While we do not have performance numbers for an *in transit* scenario where simulation and visualization overlap, our experiments indicate that *in transit* workflows can shift memory and compute burdens out of the simulation.

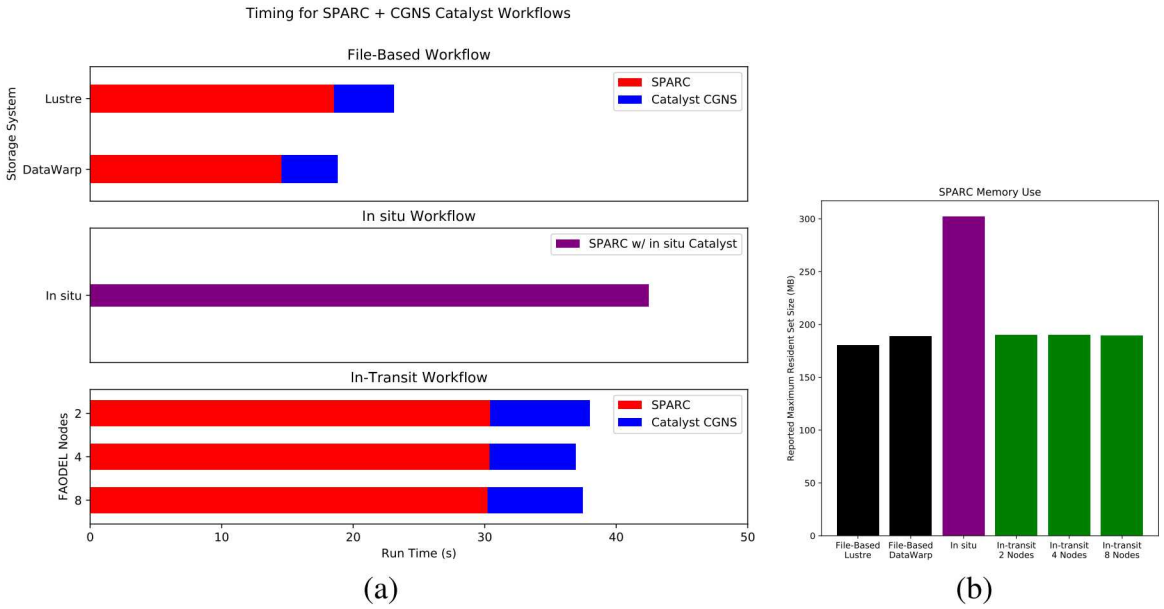

Figure 6-9. The performance measurements for (a) the time required to complete a SPARC run with CGNS-based Catalyst analysis and (b) the amount of memory required by one rank of SPARC during its execution.

7. IMPACT

The core enabling technologies for the in situ and in transit workflows are the additions to the SEACAS IOSS libraries. The *file-based* workflow used well-established IOSS tools for CGNS and Exodus. Minimal Changes to SPARC enabled these new IOSS tools to be used in all workflows. The SPARC code that exercises the IOSS interface was unchanged.

7.1. Catalyst suport for IOSS

Changes to the SEACAS library were quite extensive and include concrete descendants of the abstract class `Ioss_DatabaseIO`. These support serializing mesh data into Catalyst and into FAODEL for both surface and volume data which would typically be written and Exodus and CGNS file respectively.

In addition to the above, there are several tools that use the IOSS interface to map data into Catalyst.

`cgns2catalyst`

`exo2catalyst`

`faodel2catalyst_cgns`

`faodel2catalyst_exodus`

The above tools are named after their function, converting from a concrete `Ioss::DatabaseIO` type to Catalyst. This code was written with reuse in mind as the above are thin wrappers around a core tool `IossApplication` that maps data from the IOSS interface to Catalyst.

7.1.1. FAODEL support for IOSS

Mesh data in IOSS is principally represented by the `Ioss::DatabaseIO` class, descendant of which define a specific storage format, e.g., `Iocgns::DatabaseIO` is the `Ioss::DatabaseIO` subclass for interfacing with the CGNS library. As part of this milestone, we implemented `Iofaodel::DatabaseIO`, a new subclass of `Ioss::DatabaseIO` for interfacing the FAODEL. FAODEL is fundamentally a DHT, the key abstraction that it presents to its users is a mapping between *keys* and *blobs*, *cf.* Chapter 3. As a result, the principal function of `Iofaodel::DatabaseIO` is to map mesh data managed by IOSS into this format.

FAODEL provides support for a two-dimensional key space, with each key consisting of a row and a column. The row identifies the process that generated the key. The column of the key

uniquely identifies the data that is being stored and describes its relationship to other IOSS objects in the hierarchy.

In IOSS, we created a unique name for each Property and Field by identifying its role in the hierarchy and identifying the `State` the data belongs to. The `State` identifies a specific time-step; time-invariant data is assigned a `State` of -1. The structure of keys for Properties and Fields are shown below.

```
/State/<STATE>/Entity/<ENTITY_TYPE>/Name/<ENTITY_NAME>/Property/BasicType/<TYPE>/Name/<NAME>
```

```
/State/<STATE>/Entity/<ENTITY_TYPE>/Name/<ENTITY_NAME>/Field-
    /RoleType/<ROLE>/BasicType/<TYPE>/Name/<NAME>
```

where:

- `STATE` is an integer, $1..N$ indicating which output set it belongs to, or -1 if it is not time-dependent.
- `ENTITY_TYPE` is the type of the geometric entity, e.g., `NodeBlock` or `StructuredBlock` that "owns" the data.
- `ENTITY_NAME` is the name of the above and is often specific to the application that created is. For example, Exodus data usually has a `NodeBlock` named "nodeblock_1"
- `ROLE` can be `MESH`, `TRANSIENT`, and others that describe how the simulation uses that piece of data. `MESH` data doesn't change over time whereas `TRANSIENT` data does. The other `ROLE` type have a similar implication.
- `TYPE` is the basic datatype of the data stored by the field, e.g., `REAL`, `INT64`, etc.
- `NAME` is the name of the field or property

While this naming scheme handles the bulk of the mesh data stored by FAODEL, there are a couple of cases that have to be treated specially. First, for an `IOSS::GroupingEntity` of type `Ioss::Region`, the name assigned to the object is not independently meaningful; the name mey be used to distinguish between `Ioss::Region` objects within a single execution contenxt, but it is not guaranteed to be unique across execution contexts. As a result, the `/Name/<ENTITY>_NAME` substring is omitted from keys that identify fields or properties that belong to GroupingEntities of type `REGION`. The second special case, is data that is necessary to properly represent the mesh but is not captured in either the fields or properties of a GroupingEntity. For example, GroupingEntities of type `STRUCTUREDBLOCK` have several attributes that are not represented in their fields or properties.

7.1.2. `io_shell`

IOSS includes `io_shell`, a tool for manipulating mesh data in multiple formats (e.g., CGNS, Exodus). A common use-case for `io_shell` is to copy mesh data stored in one format to another format (e.g., copy the contents of a CGNS file to an Exodus file). As part of this milestone, we extended `io_shell` to support mesh data stored by FAODEL. By adding support for FAODEL to `io_shell`, we were able to begin to test our ability to store and retrieve mesh data in FAODEL while the integration with SPARC and Catalyst/ParaView was still under development.

As part of this milestone, we also added an entirely new function to `io_shell`: the ability to compare the mesh data stored in two different formats. By providing a new command-line option (`-compare`) to the invocation of `io_shell`, the user could verify that two sets of mesh represented the same underlying mesh. In addition to being able to copy data from existing mesh files into FAODEL, this functionality allowed us to ensure that the FAODEL representation of the mesh was equivalent to the original mesh file. Early testing using this new feature of `io_shell` was critical to our identification many errors and gaps in the original implementation of `Iofaodel::DatabaseIO`.

7.2. Extensions to SPARC

Changes to SPARC were minimal and limited to a few classes and a schema files. These changes served to allow SPARC's volume and surface writer to create concrete versions of the abstract class `Ioss_DatabaseIO`. The machinery to write mesh data to IOSS wasn't changed.

7.3. Catalyst enhancements

Catalyst was enhanced to use IOSS instead of the previous Catalyst-direct interface which require a bit more modifications to SPARC including a new Class in the hierarchy. The IOSS-based Catalyst was delivered ahead of schedule due to its association with this L2 milestone work.

7.4. Sandia Analysis Workbench (SAW)

The Sandia Analysis Workbench (SAW) [7] is a tool used internally at Sandia National Laboratories. It allows users to construct and manage workflows using graphical tools. SAW has significantly simplified the process of running important modeling and simulation tasks in production environments at Sandia. However, it is perhaps less well-suited to the dynamic and rapidly-evolving software environment found in research projects such as the one described in this report. Nonetheless, as part of this milestone, we constructed three different proof-of-concept workflows for visualizing data generated by SPARC: *in situ*, file-based, and in-transit. We have successfully demonstrated the operation of all three workflows running SPARC on Eclipse. Although our initial results ran each of the workflow components on a single rank, we are

confident, given the functionality that we have demonstrated elsewhere in this report, that there are no significant obstacles to running at larger scale in the future.

The workflow diagrams from SAW are shown in Figures 7-1 (*in situ*), 7-2 (file-based), and 7-3 (in transit). One of the challenges that we encountered with the in transit SAW workflow was that it requires FAODEL to be started as an independent process before SPARC starts. Similarly, the SAW workflow needs to know that Catalyst has completed before shutting down FAODEL. In this initial SAW workflow implementation, input from the user is required to ensure that FAODEL has successfully started and that it can be safely shut down. In the future, we plan to eliminate these two points of user interaction by exploiting existing functionality in FAODEL.

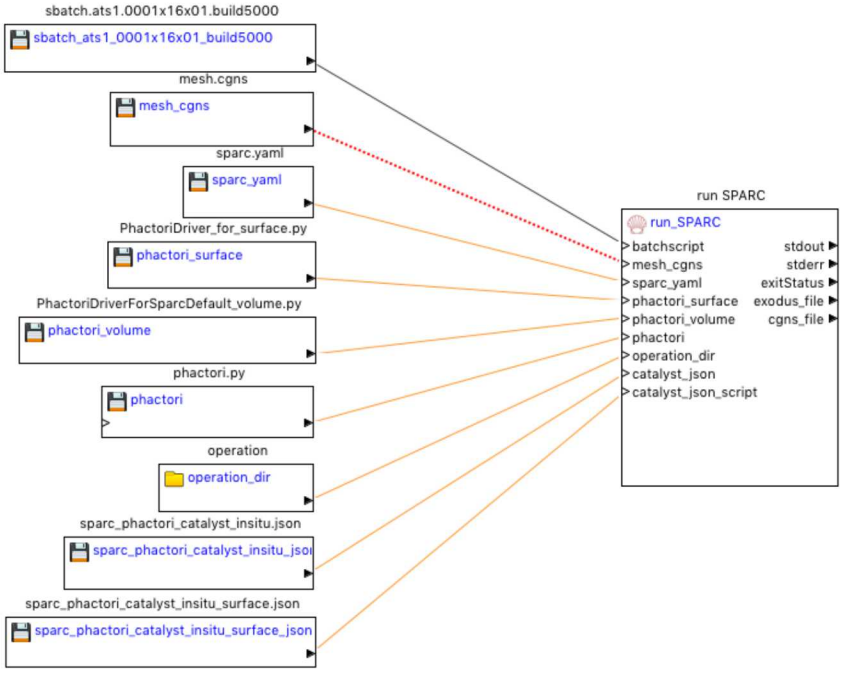

Figure 7-1. SAW workflow for running SPARC-Catalyst *in situ*

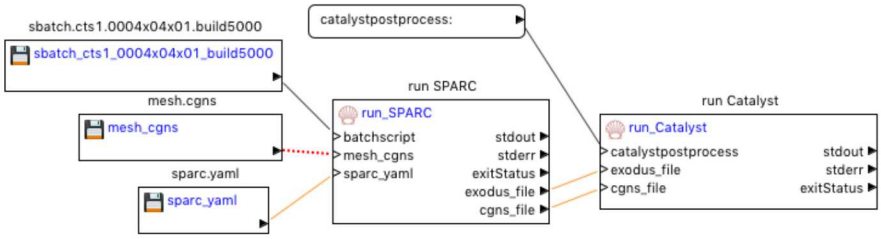

Figure 7-2. SAW workflow for running file-based SPARC-Catalyst

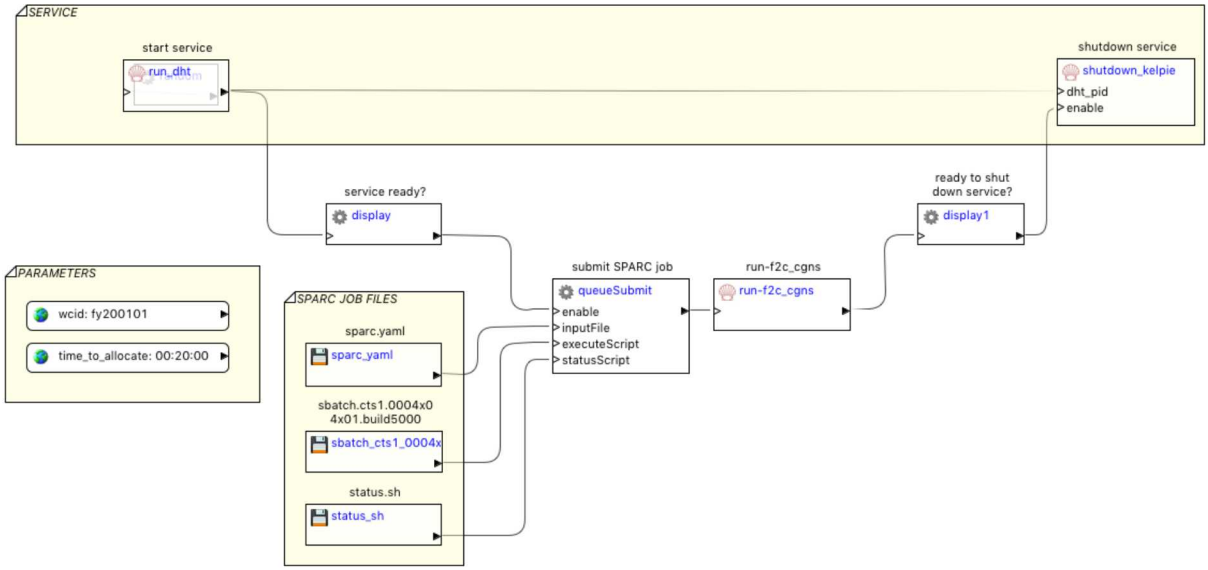

Figure 7-3. SAW workflow for running SPARC-Catalyst in transit

7.5. Results and Lessons Learned

A main goal of workflow research is to make the development and deployment of the core simulation an easier and more stable process while still enabling a variety of post-processing tools. There are a few takeaways that highlight the impact each workflow has for SPARC.

- In situ is the easiest workflow to run considering the complete workflow is launched as a single MPI job. However there is a hidden development cost that impacts SPARC developers directly. In addition to the memory trade offs detailed in the previous section, the build for in situ is considerable more complex that for either file based or in transit workflows due to the time and memory required to load in situ libraries. In particular, the Catalyst/ParaView libraries can be quite large as they were designed to be part of a stand alone application. Other in situ tools may be more amenable to minimizing library size.
- File based workflows are a bit more complex as they do require a two step process to run to completion, data generation and data post-processing. But this cost has largely been accounted for in that there are many tools designed for this workflow. So while the complete workflow is slightly more complex to run, the development process will have examples which to follow and tools that can be extended for new analysis algorthims.
- In transit workflows have a minimal impact on the simulation development teams if IOSS is already being used for mesh I/O. As described previously, there may be some work to enable a particular IOSS data management layer. However the bulk of the development complexity is squarely in the domain of the post-processing development team. As an added benefit, any instability induced by the post-processing code does not impact the execution of the simulation as FAODEL servers as a data mediator insulating the core simulation. Lastly, in transit workflows offer the opportunity for interactivity between the

simulation and post-processing. This would require some work on the development team but may pay off if the workflow can be made more efficient overall.

While it's clear that IOSS-enable applications can be used in the workflows described above, there are examples of extant codes that still use the Exodus or CGNS API directly. In these cases it's difficult to engage anything other than file based workflows without significant changes to the code base. In lieu of refactoring the codes to use IOSS, it should be possible to develop a data mediator to *replay* the Exodus and CGNS data written to disk in a way that engages in transit workflow machinery. This is in line with the Catalyst driver tools developed for this work and is a direct analogy to the second half of the in transit workflow.

8. FUTURE WORK

This milestone provided an opportunity to highlight the utility of *in transit* workflows and data services for applications of importance to the ASC program. Since *in transit* workflows and data services are not currently used in production, there was an expectation that creating a workflow for a "real" application would expose weaknesses and limitations in both our software and the HPC systems. In some cases, we were able to address these issues through clever software engineering; however, a long-term solution for production computing requires an evolution in our HPC systems to better support the needs of these types of workflows. In this chapter, we describe a strategic plan for Sandia's CSSE program to both improve our computing environment to support the use of data-services in workflows; expand the role of data services to support the anticipated needs for ASC applications, high-performance data analytics and heterogeneous platforms; and finally to collaborate with others to increase our potential impact in the HPC community.

8.1. Evolving the Computing Environment to Support Data Services

One of CSSE's objectives as written in the FY20 Implementation Plan is to "invest in and consequently influence the evolution of the computational environment." We see support for data services as a necessary evolution to enable more dynamic application workflows, integration of high-performance data analytics, and efficient utilization of heterogeneous platforms – all capabilities expected for future generations of high-performance computing deployments. Here we discuss R&D that could drive the community toward a more accommodating environment.

8.1.1. Dynamic Resource Management

For more than two decades, distributed-memory HPC systems have provided a static resource-allocation model that requires the user to explicitly allocate and manage compute-node resources using a batch scheduler. Workflow tools like SAW work with schedulers to deploy jobs within a workflow, but have to develop clever software-engineering solutions to deal with complex policies and security models. In this milestone, we used a fixed allocation of nodes for FAODEL, but the ability to spawn, grow, and shrink resources to satisfy application demand would enable a much more efficient usage for data services. In addition, our long-term vision for data services goes far beyond data management to include, for example, services for real-time system monitoring, enhanced visualization, on-demand analysis, high-performance data analytics, and others. A more dynamic resource-management approach could also enable more efficient utilization of HPC platforms.

The work to explore more dynamic resource management schemes to support data services requires a coordinated co-design effort between a number of CSSE teams including system software, I/O, Visualization and Analysis and our vendor partners (e.g., HPE, IBM, Intel, NVIDIA). Preliminary research to explore different approaches is still quite far from being production-ready, but we have initiated some work with organizations in our production computing group to better understand HPC workloads to see if approaches like oversubscribing HPC resources could enable a more flexible allocation model on dedicated partitions of an HPC system.

8.1.2. Security Models Designed to Support Data Services

HPC systems have adopted security models that often hinder the ability for independent jobs, like applications and data services, to communicate. While there are well-defined protocols for this type of communication in other environments (e.g., TCP/IP), the proprietary interconnects of our HPC vendors do not have a standard solution because there was little demand for this type of communication from other customers. Since the Cielo system at Sandia, we have had specific language in our procurement contracts to ensure vendors enable this type of communication, but the solution has been somewhat obscure an unsatisfactory. For example, on Mutrino, we had to acquire a credential from a running job, then start the next job with the same credential using a not-so-well known command tacked into our batch scheduler. We experienced issues of credentials expiring and general "bugginess" in their implementation that made generating data for this milestone a challenge.

To make data-services a viable option for production workflows, we need to solve this issue. We believe this is not a research issue, it is an engineering and design issue. We need to work with the community of laboratory researchers and HPC vendors to adopt a common solution. Now that there is growing interest service-based approaches (see Section 8.3), we believe a common solution is achievable.

8.1.3. Enhancing Workflow Tools to Support Data Services

Next Generation Workflows has made tremendous progress over the last several years, and although it has become an effective production tool, it is still missing some capability that would make it a powerful vehicle for workflows leveraging data services. First, the workflows, by design assume a somewhat serial execution model where upstream components complete before executing downstream portions of the workflow. Our vision for data services is for a more interactive relationship between the components. We require concurrent execution of the workflow components that is not easy to initiate in current IWF tools, in part because of the limitations in resource management mentioned in Section 8.1.1. Our data analysis and visualization team is already actively engaged and productively contributing to NGW. Our I/O team is involved in working with NGW on a portable path forward for data management and we plan to work with NGW on adapting the tools to enable the effective use of data services in application workflows. In the same way that we see IOSS as a key enabler and vehicle to impact

ASC codes, NGW will be a critical partner in promoting and enabling data services as a viable solution in the future.

8.1.4. *Containerized workflows for* in situ *visualization*

Container-based technology has become an increasingly important part of HPC systems. Containers provide many of the benefits of traditional (i.e., hypervisor-based) virtualization (e.g., isolation, portability) but with much lower installation, execution, and maintenance overheads. In the context of workflows, two new container-based tools have recently emerged to facilitate the deployment of *in situ* workflows.

`BeeFlow` [2] is built on the Build and Execution Environment (`BEE` [5]). `BeeFlow` is a workspace management system that manages workflows that include explicit dependencies created by *in situ* analysis tasks.[1] `BeeFlow` provides coordination to enable simulation and analysis tasks to exchange data but does not itself provide any mechanism for exchanging data between tasks. As a result, the functionality provided by `BeeFlow` is largely orthogonal to the functionality provided by the combination of FAODEL and IOSS. However, it may be possible to combine these two approaches. Although the dataflow design of `BeeFlow` does not currently support FAODEL-based dependencies, it would be possible, in principle, to modify the dataflow design of `BeeFlow` to support workflows that include FAODEL.

Containers have also been used to simplify and accelerate the process of building and distributing *in situ* and *in transit* workflows. Shudler et al. [23] demonstrated that the combination of Spack and Singularity may make it easier to manage containers that include *in situ* (or *in transit*) analysis using SENSEI. In principle, this approach could be extended to handle containers that include FAODEL.

8.2. Expanding the Role of Data Services and In Transit Analysis

A second, somewhat obvious, area for future work is to promote the proliferation and expansion of data services and *in transit* approaches. The most practical means to achieve this is to focus on application developers who can benefit from Data Services and *in transit* workflows.

8.2.1. *IOSS as a Vehicle for Delivery of R&D Capability*

Perhaps the most valuable outcome of the ATDM I/O project was a well-defined I/O-support activity that enabled our R&D team to engage directly with teams from ASC Integrated Codes. CSSE has now taken over this support role which includes explicit support for IOSS. Given the ubiquity of the IOSS mesh interface and its flexibility in writing to disparate types of data stores, the Catalyst/ParaView tool demonstrated in this milestone, could be replaced with any number of

[1] Chen et al. [2] only discuss in situ analysis; the make no mention of *in transit* analysis. Nonetheless, it does not appear that anything in their design would preclude its use for *in transit* workflows.

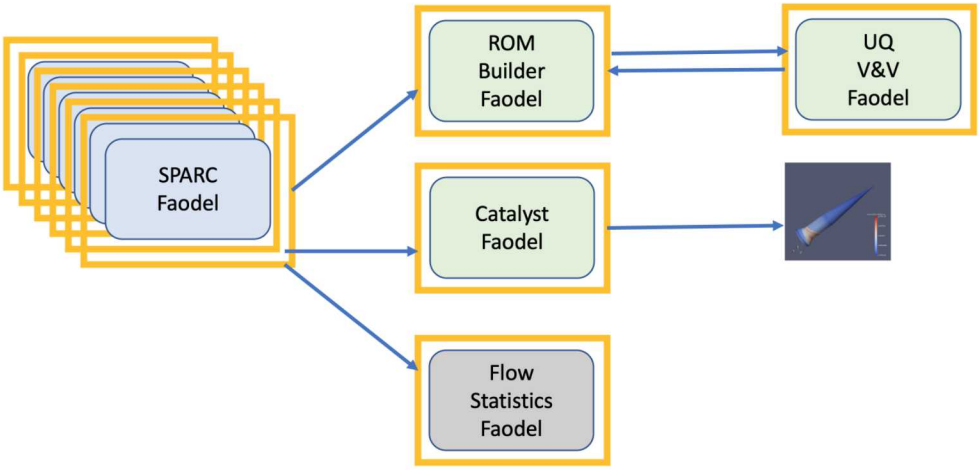

Figure 8-1. An *in transit* workflow with multiple data consumers

post-processing applications and data services that also use the IOSS interface. With improvements and extensions, this work will serve as a basis for new workflows with a number of core data producers and consumers.

8.2.2. Data Services to Enable Effective Utilization of Heterogeneous Platforms

HPC systems are becoming more heterogeneous through incorporation of various configurations of non-volatile memory, smart networks, and emerging non-Von Neumann hardware for AI, ML [21, 6], and scientific computing [22], and others. We plan to explore data services as a convenient data-management interface and performance-portability layer between conventional HPC applications and these components. In addition, there are opportunities to leverage some of these devices for the deployment of data services. For example, in a recently funded ASCR proposal, members of this team are planning to evaluate SmartNIC technology for FAODEL services. This ASCR work will build on some of the work coming out of this L2 milestone and our CSSE program expects to leverage and perhaps supplement their work by exploring other ways to leverage SmartNICs for system software, networking, and I/O.

8.2.3. SPARC ROM Building

A specific follow-on activity of this L2 milestone is an emerging collaboration with Jaideep Ray and Kenny Chowdhary to implement their ROM building for UQ and V&V work. This is particularly interesting as they have experienced problems where SPARC runs from their parameter study overwhelms the file system requiring rerunning many of their runs. The result is that a ROM-building activity that should take a few days ends up taking a couple weeks to run to completion. In addition, Greg Weirs is interested in using the in transit workflow to compute statistics of flow fields computed by SPARC.

8.3. Integration with other software

Our work is not happening in a vacuum. In this section we describe other software tools which have the potential to play complementary roles as we evolve the data services vision.

8.3.1. Enhancing Internal Workflow Tools to Support Data Services

Next Generation Workflows has made tremendous progress over the last several years, and although it has become an effective production tool, it is still missing some capability that would make it a powerful vehicle for workflows leveraging data services. First, the workflows, by design assume a somewhat serial execution model where upstream components complete before executing downstream portions of the workflow. Our vision for data services is for a more interactive and dynamic relationship between components. We require concurrent execution of the workflow components that is not easy to initiate in current IWF tools, in part because of the limitations in resource management mentioned in Section 8.1.1.

Sandia data analysis and visualization researchers are already actively engaged and productively contributing to NGW. Similarly, SNL I/O researchers are involved in working with NGW on a portable path forward for data management. We plan to work with NGW on adapting the tools to enable the effective use of data services in application workflows. In the same way that we see IOSS as a key enabler and vehicle to impact ASC codes, NGW will be a critical partner in promoting and enabling data services as a viable solution in the future.

8.3.2. Interoperating with External Projects

The problems addressed by the types of workflows explored in this milestone are not unique to the Laboratories. Other computing research and development groups within DOE are actively working to develop solutions which have reached varying degrees of maturity and deployment. Several of these projects represent potential opportunities for future collaboration:

- The Software Technology portfolio of the **Exascale Computing Project** [17] contains several products focused on data and visualization which are of potential interest. *VTK-m* is most closely related to the software already developed, adding shared-memory parallelism to the Paraview software suite. *ADIOS* provides code coupling and I/O support for applications whose needs span the storage hierarchy; a potential integration for ADIOS might specify a FAODEL service as a data destination, where that data might then be extracted using the IOSS interfaces developed for this project. *ExaIO*'s objective is a set of parallel I/O libraries based on the HDF5 container format which are storage hierarchy-aware. As FAODEL already has the ability to persist its data using HDF5, an integration with ExaIO could potentially prove useful. Lastly, the *DataLib* collection of I/O and middleware tools is similar in objective and scope to FAODEL and should be considered as a useful comparator for both functionality and performance use cases.

- **ALPINE/Ascent** [8] is an infrastructure developed by Lawrence Livermore National Laboratory that provides tools for defining the inputs, outputs, and data flow of *in situ* computations on HPC platforms. Ascent translates declarative specifications of data flow and actions into run-time *in situ* computational steps. It performs a role similar to the Catalyst software used in this project. It also provides functionality similar to the SAW tools, but with the ability to define data flow graphs for *in situ* computations as opposed to *in transit* ones. As such, it may be possible to integrate the data flow graphs used by Ascent with either the *in situ* or *in transit* FAODEL distributed hash table for data access.
- **Conduit** & **Mesh Blueprint** [9] are tools also developed at LLNL which provide facilities for definition and interchange of scientific data. Conduit features methods for defining and encapsulating an hierarchy of data objects for in-memory use or for transport across address spaces. Blueprint leverages the capabilities of Conduit to provide an interface for defining computational meshes. These tools are part of the same ecosystem as the ALPINE and Ascent software described above. Blueprint defines a computational mesh data model functionally similar to that of IOSS, which is itself historically based on the Exodus mesh file format. Several potential integrations with the FAODEL and IOSS software stacks could be explored, for examples:
  - FAODEL could be modified to add the ability to output a Conduit Node which conforms to the Mesh Blueprint protocol, providing interoperability in either *in situ* or *in transit* workflows with the Conduit ecosystem.
  - FAODEL currently performs serialization for its data transfers in an ad hoc manner without strong support for hierarchical data. The Conduit tools could be of use here.
- **SENSEI** [1] is an *in situ* execution framework which has similar goals to ALPINE and its related tools, but forms a separate software stack. SENSEI provides similar facilities for introducing *in situ* processing into an application and managing the data items used by that processing. Notably, SENSEI uses the VTK data model and can interface directly with Paraview/Catalyst while eliminating many data copying overheads. Of probably more potential integration interest, SENSEI uses ADIOS as a means of abstracting the destination of data after *in situ* processing (potentially sending data to another compute node). Integrating FAODEL as an ADIOS data route could make possible multi-stage workflows in which SENSEI *in situ* components are connected to FAODEL-based *in transit* ones.

9. CONCLUSIONS

SPARC's use of the IOSS interface provides the ability to switch between workflows with ease. This also allows data analyst developers to develop and test algorithms with a running SPARC simulation without changing the SPARC code. This is important for codes that have gone through V&V where any change could invalidate the V&V certification process.

Workflows vary by how they impact a running simulation in terms of sharing resources on local nodes. For Catalyst there is no conflict with the graphics hardware since Catalyst doesn't use the graphics cards on SNL HPC systems. This may change if VTKm is deployed in the Catalyst/Paraview dyad. It's obvious that this won't be the case for other Data Analysis tools. When those become part of this ongoing study further performance testing will help in understanding the implications of sharing graphics hardware. The *in situ* case clearly has a direct impact on the simulation processes. It may be that the bottleneck in the *in situ* workflow is in the time or memory it takes to process a chunk of output data. These chunks may be field data from a time step. Another delay we observe in Catalyst is the time and communication required for parallel rendering.

Candidates for post-processing include anything that works with output data but isn't required for the running application. There may be cases where developers want a post-processing algorithm to become part of the physics simulation. In those cases developing and testing that particular algorithm in one of these workflows can minimize the changes to the applications. Once perfected the algorithm can be refactored into the simulation's code base where V&V processes can be re-run.

The different workflow types offer tradeoffs in how simulation and post-processing tools share resources:

- In file-based workflows, the resources of interest are I/O bandwidth, file system disk space, and to a lesser extent imposed load on the parallel file system itself. While an application is not directly affected by disk space allocation, shared parallel disk arrays must continually be managed to ensure space is available. Applications which open and close many files can place abnormal load on file system metadata services, which frequently do not scale as readily as their bulk data transfer operations. Contention for I/O bandwidth is typically the main disadvantage of file-based workflows, as shared parallel file systems have only so many paths to disk. This contention can manifest as delay in completion of write operations in the application. However, this may be preferable if the parallel file system is not heavily shared. Finally, the resources used by separate later analysis steps must be considered.
- *in situ* workflows primarily address I/O contention by reducing the amount and duration of I/O write phases. The downside of this tradeoff is realized as increased application CPU usage (to perform analysis activities within application processes), increased

time-to-completion for the application, and typically increased memory usage by analysis data structures. Not all post-processing tasks are suitable for *in situ* execution, and those which are performed *in situ* can pose software engineering and maintenance problems for application developers.

- *In transit* workflows represent a tradeoff of increased complexity vs. added flexibility. Using a data management service such as FAODEL avoids both the penalties of filesystem I/O as well as the resource usage and software coupling concerns associated with *in situ* workflows. This project has demonstrated that there are significant potential benefits available to applications. This project has also demonstrated that the integration effort associated with *in transit* workflows can also be significant, and that performance evaluations and tuning are important to fully realize their advantages. Other resource considerations include the need for extra computational resources for *in transit* data management and post-processing jobs, as well as the need to coordinate their scheduling with applications. *In transit* workflows also introduce the possibility of varying the degree of asynchrony present in the workflow according to resource availability or other concerns.

Lastly, a tool like SAW can be of great benefit, coordinating and managing the complexities of an *in transit* workflow comprised of multiple data sources and consumers. An example of this is the case of building Reduced Order Models (ROM). Here a large number of simulations produce data gathered to build a ROM which can then be used in parameter studies for UQ and V&V analysis.

APPENDIX A. Build and Execution Process

A.1. Repositories

During the development of the milestone, the team used multiple git repositories across multiple projects. Table A-1 details the projects, the repository's location and the final set of commit hashes used for the experiments present in this report. In addition, each repository has been tagged with `FY20_ASC_L2_Milestone_7186` as a permament marker.

A.2. Building the Milestone Projects

Maintaining builds across all the required projects quickly became a tedious and difficult task that distracted the team from doing the actual work of the milestone. In an effort to reduce the workload on the individual team members, a repository of build scripts was developed to coordinate the build. At the highest level are superbuild scripts that clone, configure and build all the projects. For each project, a well-known branch or commit hash is used that gives developers a consistent point from which to work. Alternatively, developers can quickly change one or more of the project branches before starting a new build.

A detailed description of the build process using the superbuild scripts can be found in the DSVA/building repository located at `https://cee-gitlab.sandia.gov:/dsva/building`.

A.3. Running the Milestone Projects

The milestone focused on the HIFIRE-1 simulation using a common configuration with the only changes being the post-processing driver.

A.3.1. Running the in-situ case

The typical HIFIRE-1 configuration uses `exodus` for the surfaces and `cgns` for the volumes. The Catalyst driver is selected by prefixing either with `catalyst-`. In addition, a Catalyst script is specified as a property in the input deck which drives the visualization.

Prior to launch the environment must be configured so the Catalyst driver can find its required components are that available as shared libraries or python modules.

```
export CATALYST_ADAPTER_INSTALL_DIR=$TRILINOS_BUILD_ROOT/../install-seacas-plugin-ats1-hsw_shared
export PYTHONPATH=$PARAVIEW_ROOT/lib/python3.7/site-packages/_paraview.zip:$PYTHONPATH
SPARC_EXEC=$SPARC_BUILD_ROOT/bin/sparc
```

With this configuration and environment in place, runing the HIFIRE-1 simulation is launched the same as any SPARc job.

```
srun $SPARC_EXEC -i sparc-catalyst.yaml
```

At each timestep when the results are written, the Catalyst driver executes the visualization script which produces images.

A.3.2. Running the in-transit case

Running the in-transit case is very similar to the in-situ case. The input deck is modified to select the FAODEL driver by prefixing `exodus` or `cgns` with `faodel-` and the environment is setup in a similar way. In addition to the Catalyst driver, the FAODEL configuration information must also be specfied.

```
export CATALYST_ADAPTER_INSTALL_DIR=$TRILINOS_BUILD_ROOT/../install-seacas-plugin-ats1-hsw_shared
export PYTHONPATH=$PARAVIEW_ROOT/lib/python3.7/site-packages/_paraview.zip:$PYTHONPATH
export FAODEL_CONFIG=mutrino-sparc-iom.conf
export FAODEL_RESOURCE_URI=/ioss/dht
SPARC_EXEC=$SPARC_BUILD_ROOT/bin/sparc
```

Prior to launching the SPARC simulation, the FAODEL DHT service must be started. The DHT is mostly self-contained and launched using a play script that configures and executes the required FAODEL components. As part of the startup, a Directory Manager is created that acts as a service registry where SPARC can find the DHT.

```
srun $FAODEL_INSTALL/bin/faodel play ioss-dht.play
```

Next the simulation is launched as before.

```
srun $SPARC_EXEC -i sparc-catalyst.yaml
```

At each timestep when the results are written, the FAODEL driver transfers the data to the FAODEL DHT where it is held in memory.

When the simulation is complete, the FAODEL DHT continues to run and the faodel2catalyst_exodus or faodel2catalyst_cgns tools can be executed to pull data from the DHT and copy it to the same Catalyst plugin used by the driver in the in-situ case. Using the same visualization script as in the in-situ case, the faodel2catalyst tools generate images for each timestep written to the DHT.

Additional information about how to configure and run the experiments described in Section 6 can be found in the README file in the experiment data repository, located at `https://cee-gitlab.sandia.gov/dsva/fy20-l2-experiments`.

Project	Sub-project	Repository URL	Final Commit Hash
DSVA	building	https://cee-gitlab.sandia.gov:/dsva/building	2064308c66cd5db742c01e6513d8cd1b93206b53
	seacas	https://cee-gitlab.sandia.gov:/dsva/seacas	9d6806f965167e65bb7269a6dffea91fef02567d
	fy20-l2-experiments	https://cee-gitlab.sandia.gov:/dsva/fy20-l2-experiments	d4cbdb60a48eac0e183b934a66ed63e7101306e8
FAODEL	faodel	https://gitlab.sandia.gov:/faodel/faodel	d2d463c5abb0c3aea4e7989e314d71eb6d063e61
SPARC	sparc	https://cee-gitlab.sandia.gov:/sparc/sparc	9158a72d5dfe22b30f35d8be3ea51378f39ab678
	ear99	https://cee-gitlab.sandia.gov:/sparc/sparc-ear99	7f5baf5e55013841e818f964e8e0ce703aa702ef
	itar	https://cee-gitlab.sandia.gov:/sparc/sparc-itar	820c937e36efe87c43aeb5b75f1162cda072b36b
	performance-ear99	https://cee-gitlab.sandia.gov:/sparc/sparc-performance-ear99	e57fd23a110dfc7600875e92dbbaf6a2447c6a40
	performance-itar	https://cee-gitlab.sandia.gov:/sparc/sparc-performance-itar	6a2ccda6096dd47388b0e27800e54a742f79a24a
	regression-ear99	https://cee-gitlab.sandia.gov:/sparc/sparc-regression-ear99	30126d98a0beb29a2454e0eab4fa9f26bd39577b
	regression-itar	https://cee-gitlab.sandia.gov:/sparc/sparc-regression-itar	4651402970fb712ee62b3d6c0f809eee70b99bea
	verification-ear99	https://cee-gitlab.sandia.gov:/sparc/sparc-verification-ear99	8ca46f571cc0834f8954f3f924be06fefb4bea99
	verification-itar	https://cee-gitlab.sandia.gov:/sparc/sparc-verification-itar	4a83ae70df03e2e235c61cffe320d6d4b66f3351
	pressio	https://cee-gitlab.sandia.gov:/sparc/pressio	2900b58a395dcf85843d99d3e35e6bbd7eb11553
	trilinos	https://cee-gitlab.sandia.gov:/sparc/Trilinos	09280747ac873aa2eb1705774abace997b37e66f

Table A-1. git repositories used for ASC Milestone 7186

References

[1] U. Ayachit, B. Whitlock, M. Wolf, B. Loring, B. Geveci, D. Lonie, and E. W. Bethel. The sensei generic in situ interface. In *2016 Second Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization (ISAV)*, pages 40–44, 2016.

[2] Jieyang Chen, Qiang Guan, Zhao Zhang, Xin Liang, Louis Vernon, Allen McPherson, Li-Ta Lo, Patricia Grubel, Tim Randles, Zizhong Chen, et al. Beeflow: A workflow management system for in situ processing across hpc and cloud systems. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1029–1038. IEEE, 2018.

[3] Hank Childs, Sean D. Ahern, James Ahrens, Andrew C. Bauer, Janine Bennett, E. Wes Bethel, Peer-Timo Bremer, Eric Brugger, Joseph Cottam, Matthieu Dorier, Soumya Dutta, Jean M. Favre, Thomas Fogal, Steffen Frey, Christoph Garth, Berk Geveci, William F. Godoy, Charles D. Hansen, Cyrus Harrison, Bernd Hentschel, Joseph Insley, Chris R. Johnson, Scott Klasky, Aaron Knoll, James Kress, Matthew Larsen, Jay Lofstead, Kwan-Liu Ma, Preeti Malakar, Jeremy Meredith, Kenneth Moreland, Paul Navrátil, Patrick O'Leary, Manish Parashar, Valerio Pascucci, John Patchett, Tom Peterka, Steve Petruzza, Norbert Podhorszki, David Pugmire, Michel Rasquin, Silvio Rizzi, David H. Rogers, Sudhanshu Sane, Franz Sauer, Robert Sisneros, Han-Wei Shen, Will Usher, Rhonda Vickery, Venkatram Vishwanath, Ingo Wald, Ruonan Wang, Gunther H. Weber, Brad Whitlock, Matthew Wolf, Hongfeng Yu, and Sean B. Ziegeler. A terminology for in situ visualization and analysis systems. *The International Journal of High Performance Computing Applications*, 0(0):1094342020935991, 0.

[4] Richard L Graham, Timothy S Woodall, and Jeffrey M Squyres. Open mpi: A flexible high performance mpi. In *International Conference on Parallel Processing and Applied Mathematics*, pages 228–239. Springer, 2005.

[5] Qiang Guan. Build and execute environment. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2017.

[6] Conrad D. James, James B. Aimone, Nadine E. Miner, Craig M. Vineyard, Fredrick H. Rothganger, Kristofor D. Carlson, Samuel A. Mulder, Timothy J. Draelos, Aleksandra Faust, Matthew J. Marinella, John H. Naegle, and Steven J. Plimpton. A Historical Survey of Algorithms and Hardware Architectures for Neural-Inspired and Neuromorphic Computing Applications. *Biologically Inspired Cognitive Architectures*, 19:49–64, 2017.

[7] Sandia National Laboratories. Sandia analysis workbench (saw). `https://www.sandia.gov/saw/`, 2020. Accessed: 2020-08-25.

[8] Matthew Larsen, James Ahrens, Utkarsh Ayachit, Eric Brugger, Hank Childs, Berk Geveci, and Cyrus Harrison. The alpine in situ infrastructure: Ascending from the ashes of strawman. In *Proceedings of the In Situ Infrastructures on Enabling Extreme-Scale Analysis and Visualization*, ISAV'17, page 42–46, New York, NY, USA, 2017. Association for Computing Machinery.

[9] Lawrence Livermore National Laboratory. Conduit: Simplified Data Exchange for HPC Simulations. `https://llnl-conduit.readthedocs.io/en/latest/`. Retrieved 08-31-2020.

[10] Scott Levy, Patrick Widener, Craig Ulmer, and Todd Kordenbrock. The case for explicit reuse semantics for RDMA communication. In *Proc. Workshop on Scalable Networks for Advanced Computing Systems*, New Orleans, Louisiana, May 2020. IEEE.

[11] Scott L. Levy, Kurt B. Ferreira, Patrick M. Widener, Patrick G. Bridges, and Oscar H. Mondragon. How I Learned to Stop Worrying and Love In Situ Analytics: Leveraging Latent Synchronization in MPI Collective Algorithms. In *Proc. 23rd European MPI Users Group Meeting*, September 2016.

[12] Jiuxing Liu, Jiesheng Wu, and Dhabaleswar K Panda. High performance rdma-based mpi implementation over infiniband. *International Journal of Parallel Programming*, 32(3):167–198, 2004.

[13] Jay Lofstead, Jai Dayal, Ivo Jimenez, and Carlos Maltzahn. Efficient, failure resilient transactions for parallel and distributed computing. In *2014 International Workshop on Data Intensive Scalable Computing Systems*, pages 17–24. IEEE, 2014.

[14] Jay Lofstead, Jai Dayal, Karsten Schwan, and Ron Oldfield. D2t: Doubly distributed transactions for high performance and distributed computing. In *2012 IEEE International Conference on Cluster Computing*, pages 90–98. IEEE, 2012.

[15] Jay Lofstead, Scott Klasky, Karsten Schwan, Norbert Podhorszki, and Jin Chen. Flexible I/O and integration for scientific codes through the Adaptable I/O System. In *Proc. 6th International Workshop on Challenges of Large Applications in Distributed Environments*, June 2008.

[16] Jay Lofstead, Ron Oldfield, Todd Kordenbrock, and Charles Reiss. Extending scalability of collective I/O through Nessie and staging. In *Proceedings of the 6th Parallel Data Storage Workshop*, PDSW '11, pages 7–12, Seattle, WA, November 2011.

[17] P. Messina. The Exascale Computing Project. *Computing in Science & Engineering*, 19(3):63–67, 2017.

[18] A. Moody, G. Bronevetsky, K. Mohror, and B. R. d. Supinski. Design, modeling, and evaluation of a scalable multi-level checkpointing system. In *SC '10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11, 2010.

[19] Ron A. Oldfield, Patrick Widener, Arthur B. Maccabe, Lee Ward, and Todd Kordenbrock. Efficient data-movement for lightweight I/O. In *Proceedings of the 2006 International Workshop on High Performance I/O Techniques and Deployment of Very Large Scale I/O Systems*, Barcelona, Spain, September 2006.

[20] Matthew Rocklin. Dask: Parallel computation with blocked algorithms and task scheduling. In *Proceedings of the 14th python in science conference*, number 130-136 in 14. Citeseer, 2015.

[21] Catherine D. Schuman, Thomas E. Potok, Robert M. Patton, J. Douglas Birdwell, Mark E. Dean, Garrett S. Rose, and James S. Plank. A Survey of Neuromorphic Computing and Neural Networks in Hardware. *CoRR*, abs/1705.06963, 2017.

[22] W. Severa, O. Parekh, K. D. Carlson, C. D. James, and J. B. Aimone. Spiking Network Algorithms for Scientific Computing. In *2016 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8, Oct 2016.

[23] Sergei Shudler, Nicola Ferrier, Joseph Insley, Michael E Papka, and Silvio Rizzi. Spack meets singularity: creating movable in-situ analysis stacks with ease. In *Proceedings of the Workshop on In Situ Infrastructures for Enabling Extreme-Scale Analysis and Visualization*, pages 34–38, 2019.

[24] Hiroshi Tezuka, Francis O'Carroll, Atsushi Hori, and Yutaka Ishikawa. Pin-down cache: A virtual memory management technique for zero-copy communication. In *Proceedings of the First Merged International Parallel Processing Symposium and Symposium on Parallel and Distributed Processing(IPPS/SPDP) 1998*, pages 308–314. IEEE, 1998.

[25] T. P. Wadhams, E. Mundy, M. G. MacLean, and M. S. Holden. Ground test studies of the hifire-1 transition experiment part 1: Experimental results. *JOURNAL OF SPACECRAFT AND ROCKETS*, 45(6):1134–1148, 2008.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.