Scientific Workflows at DataWarp-Speed: Accelerated Data-Intensive Science using NERSC's Burst Buffer

Andrey Ovsyannikov*, Melissa Romanus†, Brian Van Straalen*, Gunther H. Weber*‡ and David Trebotich*
*Lawrence Berkeley National Laboratory, Berkeley, CA 94720
†Rutgers University, Piscataway, NJ 08854
‡University of California, Davis, CA 95616

***Abstract*—Emerging exascale systems have the ability to accelerate the time-to-discovery for scientific workflows. However, as these workflows become more complex, their generated data has grown at an unprecedented rate, making I/O constraints challenging. To address this problem advanced memory hierarchies, such as burst buffers, have been proposed as intermediate layers between the compute nodes and the parallel file system. In this paper, we utilize Cray DataWarp burst buffer coupled with in-transit processing mechanisms, to demonstrate the advantages of advanced memory hierarchies in preserving traditional coupled scientific workflows. We consider in-transit workflow which couples simulation of subsurface flows with on-the-fly flow visualization. With respect to the proposed workflow, we study the performance of the Cray DataWarp Burst Buffer and provide a comparison with the Lustre parallel file system.**

I. Introduction

Scientific workflows running at scale on high-performance computing systems play a critical role in supporting modern scientific discovery. These workflows provide a way for scientists to organize the various software and application components required to simulate naturally-occurring phenomena and to define the interactions between them. One such critical interaction, i.e., the exchange of data between the different components, has traditionally been achieved through the writing and reading of files during workflow execution. However, as simulated science systems continue to grow in size and complexity, the rate at which computations can be performed is becoming much faster than the rate at which files can be read and written. As a result, it is widely suspected that the file-based method of workflow data exchange will become untenable on next generation high-performance systems [1], [2].

Looking at hardware trends for these next-generation machines [3], one constraint that is common with today's systems is *locality*. Simulation codes will be severely hindered by insufficient bandwidth and high latency to persistent storage media. Performing data analytics *in-situ*, i.e., moving the analytics close to where the data is generated, has been proposed as a means of addressing these issues.

There are two primary ways of realizing in-situ workflows on modern supercomputers, both of which have limitations. In the first such way, data processing is incorporated with the simulation to create a unified executable. This allows the analytics component to access data generated by the simulation by making function calls to their shared memory address space. In-situ processing has garnered recent interest because it provides a way for data-intensive workflows to achieve I/O acceleration, which can reduce the overall end-to-end workflow runtime dramatically. However, this approach assumes that a file system does not provide sufficient performance and that the only way to circumvent its limitations is to make drastic changes that flatten the workflow into a single executable.

The second in-situ method attempts to relax the requirements of compiling a unified executable using data staging. Data staging enables workflow coupling by using a subset of compute nodes as dedicated I/O nodes. Current data staging frameworks for in-situ programming that employ this approach include PreDatA [4], DataStager [5], and DataSpaces [6]. Independently compiled applications can then use the memory space on these nodes to store and exchange intermediate workflow data.

However, there are drawbacks to this approach. First, the frameworks above require users to integrate a data staging API into their applications. Although the applications can still be compiled independently, they do need to be recompiled with the staging API integrated within them. Second, staging always requires extra compute resources. Lastly, these solutions have a limited scope of persistence and utilize one of the highest commodity memory resources (on-node DRAM), and exascale memory trends show that on-node DRAM is expected to shrink.

For these reasons, we assert that the traditional software engineering model still offers the greatest flexibility and requires the least software changes, if the specific flaws of file system performance can be mitigated. In this paper, we utilize a state-of-the-art Cray DataWarp burst buffer as an intermediate layer between compute nodes and a traditional parallel filesystem (composed of HDDs) to show the potential benefits of in-transit processing with advanced memory appliances to the scientific community. The relevant scientific question then is whether the Cray DataWarp burst buffer sufficiently ameliorates the lagging performance of traditional persistent storage technology and whether we have sufficient control of the HPC

resources to manage a realistic scientific workflow.

We propose an in-transit workflow which couples direct numerical simulation with on-the-fly flow visualization. With respect to proposed workflow, we study the performance of the Cray DataWarp Burst Buffer and provide a comparison with the Lustre parallel file system.

II. The Cray DataWarp Burst Buffer

We utilize a state-of-the-art Cray DataWarp burst buffer, as part of the National Energy Research Scientific Computing (NERSC) Center's Cori Phase I system. Burst buffer provides an additional layer of memory that sits in between the system compute nodes and the underlying parallel file system. It was created as a means of addressing the growing gap between computation and I/O that many workflows and applications currently experience on high-performance computing (HPC) systems.

The current Cori Phase 1 system has 144 burst buffer nodes. Each burst buffer node contains two 3.2 TB NAND flash solid-state drives (SSDs), 64GB DRAM, and two 8-core Intel Xeon processors. Burst buffer nodes are attached directly to the Cray Aries network interconnect of the Cori system. Detailed NERSC DataWarp Burst Buffer hardware specification is given in [7].

In the Cray DataWarp burst buffer implementation, the SSDs are exposed to applications as a file system. This offers the application developer a variety of options for file I/O and coordination. It also does not require code changes for file reading and writing, except to change the path from Lustre to the burst buffer. Consequently, HPC codebases that are already integrated with high-volume data management tools, such as HDF5 [8] or ADIOS [9], are also able to easily use the burst buffer.

The access to burst buffer resources is managed via the workload manager; on Cori, the installed scheduling system is Slurm [10], which has been integrated with the Cray DataWarp API. DataWarp resources are requested in a Slurm script using the `#DW` directive. This directive can also be used to stage a file from the PFS to the burst buffer or stage a file out from the burst buffer to PFS. Both of the staging operations are asynchronous and can occur multiple times within the batch script. Slurm will execute stage-in operations prior to the job start-up time. Users can request burst buffer reservations that last for the length of a job or persist until the user releases it or the enforced limit is reached.

Documented burst buffer use cases include checkpoint/restart, staging (for pre-fetching or post-processing), out-of-core, in-situ/in-transit analytics/vizualization, and I/O acceleration [11]. This work builds upon the in-transit use case and extends the concept further by illustrating the use of the burst buffer for real time workflow coupling and exchange. By utilizing the burst buffer as a medium for file exchange between independently running programs, the applications can asynchronously access data in the space as soon as it becomes available.

III. A Motivating Use Case: Carbon Sequestration & Shale Gas Extraction

The geologic subsurface constitutes the Nations primary source of energy and also provides a vast amount of storage space critical to a low-carbon and secure energy future. A major theme regarding the efficiency and security of subsurface energy utilization is an advanced predictive capability for the multiscale mechanical behavior and reactive multiphase transport in nanoporous (very low permeability) rock layers like shales. Such prediction is critical to unravel phenomena associated with unconventional oil and gas extraction or to ensure secure structural trapping of CO_2 during early phases of carbon storage.

Chombo-Crunch [12], [13] is a high performance subsurface simulator used for modeling pore scale reactive transport processes associated with carbon sequestration. It is a combination of Chombo [14] based advection and diffusion solvers and the multicomponent geochemistry module of CrunchFlow [15]. The code is ideal for modeling pore scale processes because of the embedded boundary method which treats very complex geometries created by heterogeneous porous media. This approach explicitly resolves reactive surface areas of minerals as well as achieves high resolution in diffusive boundary layers. The code has been validated by reactive transport experiments [16] and scales to full machine capability [12], [17]. It is also being applied to shale gas extraction, nuclear waste repositories, and even battery electrodes.

Performing higher resolution simulations increases the number of variables and size of the data being shared by Chombo-Crunch. The output files generated by Chombo-Crunch can vary from several GB to several TB in size per time step. In the traditional file-based execution of Chombo-Crunch using Lustre, checkpoint and output data can only be written approximately every 100 timesteps, because the I/O would otherwise dominate the computing allocation or overrun the PFS allocation. The result is simulation snapshots that are highly refined spatially, but quite coarse temporally. In this study, we explore turning the post-processing tempo to match the time scale of simulation, resulting in 100x more plot files.

IV. Design & Implementation

Figure 1 shows the Chombo-Crunch+VisIt workflow as it is currently integrated with the burst buffer. The figure highlights the utilization of the burst buffer as a coordination mechanism between workflow components. In this section, we describe the execution flow between these components.

The workflow begins with a user requesting compute and burst buffer resources via Cori's workload manager, Slurm. The resource requests and job execution logic can be in a single job script, as shown by Example Script 1. To further exercise burst buffer capabilities, we perform a `stage_in` of an HDF5 restart file generated by a previous run of Chombo-Crunch. After submission, Slurm waits until a job is about to become active in the queue and determines which compute nodes it will allocate when the job reaches its expected runtime. It then reserves the requested burst buffer space and

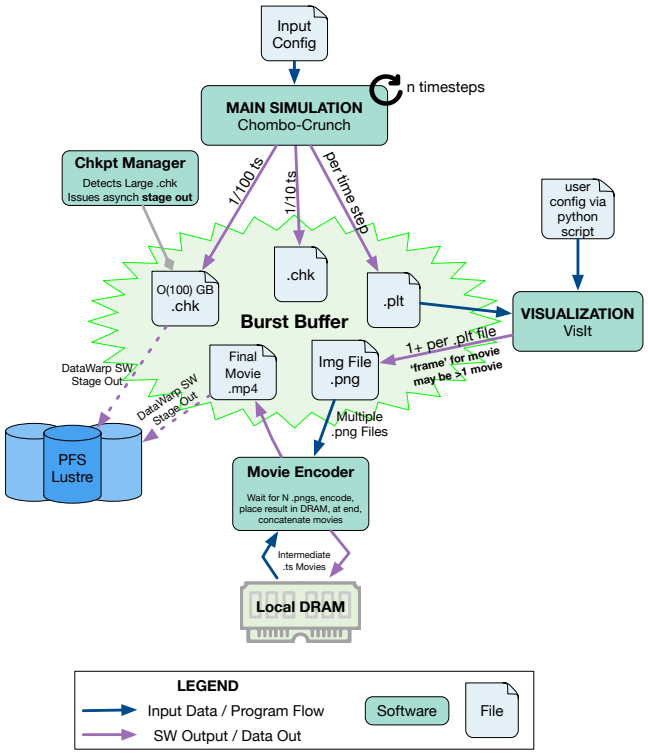

Fig. 1: Chombo-Crunch and VisIt Integrated Burst Buffer Workflow Diagram

Example Script 1: Slurm batch script with implementation of Chombo-Crunch and VisIt Integrated Workflow.

```
#!/bin/bash
#SBATCH --nodes=1040
#DW jobdw capacity=200TiB access_mode=striped type=scratch
#DW stage_in type=file source=/pfs/restart.hdf5 destination
    =$DW_JOB_STRIPED/restart.hdf5

### Load required modules
module load visit

ScratchDir="$SLURM_SUBMIT_DIR/_output.$SLURM_JOBID"
BurstBufferDir="${DW_JOB_STRIPED}"
mkdir $ScratchDir
stripe_large $ScratchDir
NumTimeSteps=2000
RestartFileName="restart.hdf5"
ProgName="chombocrunch3d.Linux.64.CC.ftn.OPTHIGH.MPI.PETSC.
    ex"
ProgArgs=chombocrunch.inputs
ProgArgs="$ProgArgs check_file=${BurstBufferDir}check
    plot_file=${BurstBufferDir}plot pfs_path_to_checkpoint=
    ${ScratchDir}/check restart_file=${BurstBufferDir}${
    RestartFileName} max_step=$NumTimeSteps"

### Launch Chombo-Crunch
srun -N 1024 -n 32768 $ProgName $ProgArgs &
### Launch VisIt
visit -l srun -nn 16 -np 512 -cli -nowin -s VisIt.py &
### Launch Encoder
./encoder.sh -pngpath $BurstBufferDir -endts $NumTimeSteps
wait

### Stage-out movie file from Burst Buffer
#DW stage_out type=file source=$DW_JOB_STRIPED/movie.mp4
    destination=/pfs/movie.mp4
```

uses the DataWarp API to stage the file into the burst buffer, thereby moving it closer to the execution. Once the job starts, Chombo-Crunch reads the restart file from the burst buffer over the high-speed network instead of the parallel file system.

The simulation writes checkpoint files at a user-specified interval. In the figure, this interval is once every ten timesteps. Writing a checkpoint every timestep would cause the simulation to spend too much time in I/O as opposed to actual calculation. At longer intervals (also user-defined), Chombo-Crunch writes a large checkpoint file with more extensive information. In order to allow the simulation to progress sooner, it writes its checkpoint file directly to the burst buffer. In the current implementation, an asynchronous DataWarp `stage_out` call is made by the simulation immediately following the file close operation. The simulation returns to processing right away, while the DataWarp service is responsible for moving the file to the slower, hard disk–based layer.

The plot file is generated once per time step. The user's python script tells the VisIt compute engine what data is relevant for generating visual information. In this workflow, the python script actively loops to check for the presence of a new plot file in the burst buffer working directory. As soon as it sees this plot file, it begins operating on the data, according to the user specifications. The output written back to the burst buffer is a PNG file that represents one movie frame. Note that VisIt can potentially output more than one PNG per time step if more than one variable is of interest.

The movie encoder waits until a certain number of PNG files is present in the burst buffer working directory and then encodes those files into a temporary .mp4 or .ts (transport stream) file. These files are written to the local DRAM where the encoder is running, since ffmpeg will utilize them again in the concatenate step.

This process continues until Chombo-Crunch has completed the objective for the current simulation. The simulation executable exits, and VisIt operates on the final plot file. The movie encoder sees the final PNG file, encodes it, and performs a concatenation of all of the intermediate movies. The final movie is then written back into the burst buffer and staged out to the PFS.

A. *Modifications Required*

The previous Chombo-Crunch+VisIt workflow, like many other scientific workflows, was based on a series of sequential stages (viz. one component would run from start to finish before the next component could begin). The exchange of data between the simulation and visualization occurred via files written to and read from the PFS. After the visualization stage was complete, the PNG frames were remotely transferred to the scientist's personal computer for offline encoding.

Since the burst buffer is also exposed as a file system, no modifications were required to the workflow communication method (i.e., via files). The simple act of integrating the burst buffer into the workflow provided improved read/write performance, due to the flash-based NVRAM technology. Additionally, communication between compute nodes and the

burst buffer over high-speed InfiniBand connections enabled faster ingestion and expulsion of data by component applications.

However, the transition from a sequential, stage-based workflow to a dynamic, asynchronous one created a new set of challenges. Previously, synchronization during runtime was not important, since all of the files were present at the beginning of a stage. However, in the modified workflow, the existence of a new file did not necessarily indicate that it was done being written to. To address this issue, Chombo-Crunch was modified to write data to a temporary file during the writing process and only change the file name after the write was finished. This exploits a *very* important feature of a POSIX-compliant file abstraction: `rename` is *atomic* [18]. Thus, VisIt could rely on the existence of a specific file name as a trigger to begin reading data. The atomicity of `rename` is one reliable notification mechanism that has a chance of being portable.

Another important consideration in the modified workflow was the rate at which Chombo-Crunch generated plot files compared to the rate at which VisIt could process them. If the Chombo-Crunch rate was much faster than the VisIt processing rate, the overlap of the two components would not result in a significant reduction in the end-to-end runtime. The experiments in this paper were carefully chosen such that VisIt was able to process the plot files at a faster rate. Although this accounts for load balancing in the workflow execution, the manual aspect, which utilized pre-existing expertise of the domain scientists and visualization experts, is not ideal. If VisIt is evaluating many variables and trying to write several different PNGs per plot file, it is likely to be slower. This motivates the need for autonomous load balancing and management of I/O for coupled workflow components. If a component (e.g., VisIt) becomes overwhelmed, the execution of the main simulation (e.g.,Chombo-Crunch) could be stopped until it is able to catch up. Alternatively, the ability to dynamically adjust the number of instances for the slower component or the opportunity to adjust the way that the cores for the job are allocated can also provide a means of load balancing. The autonomous load balancing is the subject of ongoing work and it is not presented in this paper.

This iteration of the workflow also incorporated the DataWarp API into the Chombo-Crunch code, as a means of staging out files in lieu of a checkpoint manager. However, this created a dependency between the simulation and the specific DataWarp implementation of the burst buffer. Once the logic is pulled out of the simulation and put into a separate checkpoint manager code, it will enable the complete workflow to be ported to other emerging burst buffer and deep memory hierarchy architectures with minimal changes.

V. Experimental Results

To assess the performance of the in-transit workflow utilizing DataWarp, we conducted a set of numerical simulations of chemically reactive fluid flows in complex porous media geometries. The first example is a "packed cylinder" problem [12], [13], [16], which represents a calcite dissolution process in a cylinder packed by random set of spheres. This particular benchmark has been used for Chombo-Crunch weak scaling study [17], and we will use it an an I/O bandwidth study in the present work. In addition to synthetic geometries as in the packed cylinder case, we perform direct simulation from microtomography image data. The second case study is a reactive flow in a fractured dolomite [19]. The simulations have been performed on NERSC Cori Phase I system with Intel Haswell^{TM} processors (two 2.3 GHz 16-core processors per each node). Table I summarizes the details of considered case studies.

TABLE I: **A collection of case studies for in-transit workflow.**

Benchmark	# of grid cells	# of cores	
		Chombo-Crunch	VisIt
Packed cylinder	$256^3 - 1024^3$	512-32768	32-512
Dolomite	$384 \times 160 \times 768$	512	64

A. Bandwidth scaling study

For weak scaling analysis, we perform 7 different simulations for packed cylinder problem. To keep the total number of grid cells per processing element constant, we consider various domain sizes by changing the cylinder aspect ratio from 1:1 to 4:1 and increasing total number of grid cells from 256^3 to 1024^3. The total number of cores for weak scaling study is in a range from 512 to 32768 cores. From the I/O perspective, this results to plot file size varying from 7 GiB to 472 GiB and checkpoint file size - from 6 GiB to 385 GiB. More details about settings of weak scaling benchmark can be found in [17].

Table II shows the results of the scaling study for 512 to 32768 MPI tasks for I/O to Burst Buffer and provides a comparison with I/O performance results of Lustre file system. The number of compute nodes to Burst Buffer nodes is fixed at 16:1. The number of Burst Buffer nodes is increasing from 1 to 64 for and number of compute nodes is increasing from 16 (512 cores) to 1024 (32768 cores), respectively. Results for Lustre file system have been obtained with total number of Object Storage Targets (OSTs) of 72 and a stripe size of 1 MiB in all cases which was found to give an optimal performance for current application. As it is seen from the table the Burst Buffer results are significantly better than its Lustre counterparts in all considered weak scaling test runs. Figure 2 shows that good scalability for bandwidth has been achieved both for Lustre and Burst Buffer. As it is seen, the I/O bandwidth for Burst Buffer significantly outperforms bandwidth for the Lustre file system: the achieved improvement varies from 2.84x to 5.73x depending on file size.

B. Full in-transit workflow: Simulations from image data

The next experiment is aimed at assessing the full in-transit workflow shown in Figure 1. For this purpose, Chombo-Crunch production example has been used and it is a reactive

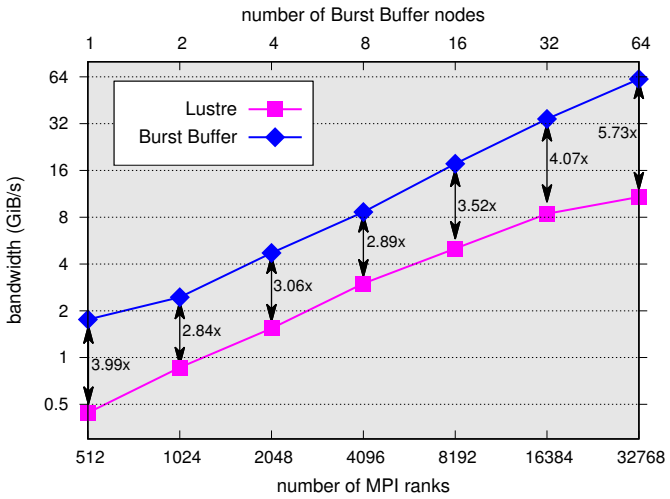

Fig. 2: I/O bandwidth weak scaling study. Shows I/O performance between Burst Buffer and Lustre with increasing number of processes writing data to a shared HDF5 plot file.

TABLE II: **I/O bandwidth weak scaling study. Shows I/O performance with increasing number of processes writing data to a HDF5 plot file. The ratio of number of compute nodes to Burst Buffer nodes is fixed at 16:1.**

# cores	File Size [GiB]	Write time [sec]		Bandwidth [GiB/sec]	
		Lustre	BB	Lustre	BB
512	7.37	16.72	4.19	0.44	1.76
1024	14.75	17.08	6.03	0.86	2.44
2048	29.5	19.17	6.26	1.54	4.71
4096	59	19.70	6.83	2.99	8.64
8192	118	23.49	6.68	5.02	17.66
16384	236	28.06	6.89	8.41	34.25
32768	472	43.68	7.62	10.8	61.94

fluid flow in fractured dolomite [19]. In this case, the geometry of porous media is obtained from microtomography image data. We discretize a computational domain with 47M grid cells and each cell size is 13 microns. 16 compute nodes with 512 cores are used by Chombo-Crunch, 2 nodes for VisIt, and 1 extra core on the login node used by the encoder. This simulation uses the Burst Buffer at full current capacity (144 nodes).

Table III shows the amount of I/O for two considered case studies. The results are summarized for different I/O intensities: (a) High intensity I/O pattern: checkpoint interval is 10 timesteps, plot file every time step; (b) Medium intensity I/O: checkpoint interval is 100 timesteps, plot file interval is 10 timesteps; (c) Low intensity I/O: checkpoint interval is 500 timesteps, plot file interval is 100 timesteps.

VI. Conclusion & Future Work

For workflows dominated by a primary computation and several ancillary post-processing steps, a file-based workflow augmented by NVRAM can make productive use of a large Cray XC40 computing platform. We designed and deployed a workflow using a bash batch script for the NERSC Cori

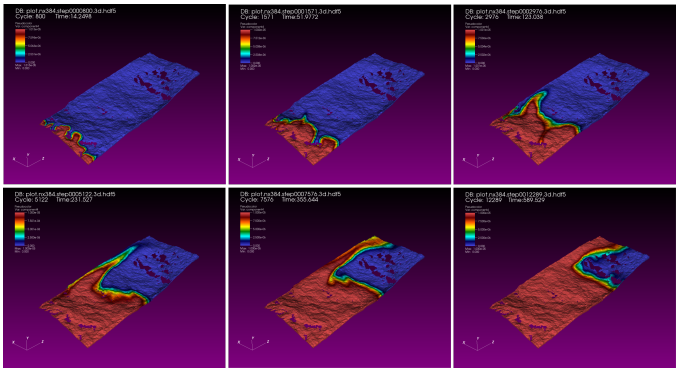

Fig. 3: Snapshots from in-transit visualization of the reactive flow in the fractured dolomite. This simulation has been performed on NERSC Cori Phase I system using 16 compute nodes (512 cores) for Chombo-Crunch, 2 compute nodes (64 cores) for VisIt and 144 Burst Buffer nodes for I/O.

TABLE III: **Details on compute and I/O time for 2 case studies. Three I/O patterns have been considered: (a) High intensity I/O pattern: checkpoint interval is 10 timesteps, plot file interval is 1 timestep; (b) Medium intensity I/O: checkpoint interval is 100 timesteps, plot file interval is 10 timesteps; (c) Low intensity I/O: checkpoint interval is 500 timesteps, plot file interval is 100 timesteps.**

	Dolomite problem		Packed cylinder	
	Lustre	BB	Lustre	BB
# of timesteps	20000		4000	
plot file size	7.46 GiB		118 GiB	
checkpoint size	6.12 GiB		96.2 GiB	
compute time per ts	9.87 s		12.19 s	
checkpoint write time	47.28 s	1.47 s	49.49 s	6.85 s
plot file write time	14.45 s	0.62 s	23.49 s	1.43 s
I/O time, pattern (a)	66%	13.8%	73.5%	17.1%
I/O time, pattern (b)	16.3%	0.77%	21.8%	2.00%
I/O time, pattern (c)	2.36%	0.126%	3.16%	0.27%

computing platform. The performance boost of an NVRAM file-system does not need to be dramatic to make it effective. The obtained results showed that the burst buffer provided a definite I/O improvement to the Chombo-Crunch+VisIt workflow and successfully reduced the overall end-to-end run time. Closing a factor of 10 in latency and bandwidth was sufficient to allow Chombo-Crunch to move to every-timestep post-processing while only changing roughly 20 lines of source code in Chombo. Neither Chombo-Crunch nor VisIt required any change to their scalable computing designs or build process. For post-processing that is embarrassing parallel (like movie frame generation), there is an easy path to exascale post-processing by launching as many VisIt parallel compute engines as is needed to keep the buffer capacity in check.

Moving forward, we plan to develop the supervisor capabilities needed to augment our simple bash script workflow to manage system variability and start work on designing signaling designs suitable for HPC platforms to enable more robust and reactive workflows within the Slurm batch management system.

REFERENCES

[1] K.-L. Ma, C. Wang, H. Yu, and A. Tikhonova, "In-situ processing and visualization for ultrascale simulations," *Journal of Physics: Conference Series*, vol. 78, no. 1, p. 012043, 2007. [Online]. Available: http://stacks.iop.org/1742-6596/78/i=1/a=012043

[2] A. Kageyama and T. Yamada, "An approach to exascale visualization: Interactive viewing of in-situ visualization," *Computer Physics Communications*, vol. 185, no. 1, pp. 79–85, 2014.

[3] P. Kogge and J. Shalf, "Exascale computing trends: Adjusting to the," *Computing in Science & Engineering*, vol. 15, no. 6, pp. 16–26, 2013.

[4] F. Zheng, H. Abbasi, C. Docan, J. Lofstead, S. Klasky, Q. Liu, M. Parashar, N. Podhorszki, K. Schwan, and M. Wolf, "PreDatA - preparatory data analytics on peta-scale machines," in *Proc. of 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS'10)*, April 2010.

[5] H. Abbasi, M. Wolf, G. Eisenhauer, S. Klasky, K. Schwan, and F. Zheng, "Datastager: scalable data staging services for petascale applications," in *Proc. 18th International Symposium on High Performance Distributed Computing (HPDC'09)*, 2009.

[6] C. Docan, M. Parashar, and S. Klasky, "DataSpaces: An Interaction and Coordination Framework for Coupled Simulation Workflows," in *Proc. of 19th International Symposium on High Performance and Distributed Computing (HPDC'10)*, June 2010.

[7] W. Bhimji, D. Bard, D. Paul, M. Romanus, A. Ovsyannikov, B. Friesen, M. Bryson, J. Correa, G. Lockwood, V. Tsulaia, S. Byna, S. Farrell, C. Daley, V. Beckner, B. V. Straalen, D. Trebotich, C. Tull, G. Weber, N. Wright, K. Antypas, and Prabhat, "Accelerating science with the NERSC Burst Buffer Early User Program," *Cray User Group Meeting*, 2016.

[8] The HDF Group. (2000-2010) Hierarchical data format version 5. [Online]. Available: http://www.hdfgroup.org/HDF5

[9] J. Lofstead, F. Zheng, S. Klasky, and K. Schwan, "Adaptable, Metadata Rich IO Methods for Portable High Performance IO," in *Proc. 23th IEEE International Parallel and Distributed Processing Symposium (IPDPS'09)*, May 2009.

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

[11] M. Romanus, R. B. Ross, and M. Parashar, "Challenges and considerations for utilizing burst buffers in high-performance computing," *CoRR*, vol. abs/1509.05492, 2015. [Online]. Available: http://arxiv.org/abs/1509.05492

[12] D. Trebotich, M. F. Adams, S. Molins, C. I. Steefel, and S. Chaopeng, "High-resolution simulation of pore-scale reactive transport processes associated with carbon sequestration," *Computing in Science & Engineering*, vol. 16, no. 6, pp. 22–31, 2014.

[13] S. Molins, D. Trebotich, C. I. Steefel, and C. Shen, "An investigation of the effect of pore scale flow on average geochemical reaction rates using direct numerical simulation," *Water Resour. Res.*, vol. 48, no. 3, pp. 43–82, 2012.

[14] M. Adams, P. Colella, D. T. Graves, J. Johnson, N. Keen, T. J. Ligocki, D. F. Martin, P. McCorquodale, D. Modiano, P. Schwartz, T. Sternberg, and B. V. Straalen, *Chombo Software Package for AMR Applications, Design Document*. Lawrence Berkeley National Laboratory Technical Report LBNL-6616E.

[15] C. I. Steefel, *CrunchFlow Users Manual*. Lawrence Berkeley National Laboratory, 2008.

[16] S. Molins, D. Trebotich, L. Yang, J. B. Ajo-Franklin, T. J. Ligocki, C. Shen, and C. Steefel, "Pore-scale controls on calcite dissolution rates from flow-through laboratory and numerical experiments," *Environmental Science and Technology*, 2014.

[17] D. Trebotich and D. Graves, "An adaptive finite volume method for the incompressible Navier–Stokes equations in complex geometries," *Communications in Applied Mathematics and Computational Science*, vol. 10, no. 1, pp. 43–82, 2015.

[18] B. Pollak, "Portable operating system interface (posix)-part 1x: Real-time distributed systems communication application program interface (api)," *IEEE Standard P*, vol. 1003.

[19] J. B. Ajo-Franklin, M. Voltolini, S. Molins, and L. Yang, "Coupled processes in a fractured reactive system: A dolomite dissolution study with relevance to gcs caprock integrity," *Caprock Integrity in Geological Carbon Storage*, Submitted December 2015. In review.