Building A Scalable Forward Flux Sampling Framework using Big Data and HPC

Ryan S. DeFever
Chemical and Biomolecular Engineering
Clemson University
Clemson, SC
rdefeve@clemson.edu

Walter Hanger
Chemical and Biomolecular Engineering
Clemson University
Clemson, SC
whanger@clemson.edu

Sapna Sarupria
Chemical and Biomolecular Engineering
Clemson University
Clemson, SC
ssarupr@clemson.edu

Jon Kilgannon
Computer Science
West Chester University
West Chester, PA
jk880380@wcupa.edu

Amy W. Apon
School of Computing
Clemson University
Clemson, SC
aapon@clemson.edu

Linh B. Ngo
Computer Science
West Chester University
West Chester, PA
lngo@wcupa.edu

ABSTRACT

Forward flux sampling (FFS) is an established scientific method for sampling rare events in molecular simulations. However, as the difficulty of the scientific problem increases, the amount of data and the number of tasks required for FFS is challenging to manage with traditional scripting tools and languages for high performance computing. The SAFFIRE software framework has been developed to address these challenges. SAFFIRE utilizes Hadoop to manage a large number of tasks and data for large scale FFS simulations. The framework is shown to be highly scalable and able to support large scale FFS simulations. This enables studies of rare events in complex molecular systems on commodity cluster computing systems.

CCS CONCEPTS

• **Computing methodologies** → **Molecular simulation**; **Rare-event simulation**; **Massively parallel and high-performance simulations**; • **Applied computing** → *Chemistry*.

KEYWORDS

Forward Flux Sampling, Hadoop, data-intensive computing, molecular simulations, rare events

ACM Reference Format:
Ryan S. DeFever, Walter Hanger, Sapna Sarupria, Jon Kilgannon, Amy W. Apon, and Linh B. Ngo. 2019. Building A Scalable Forward Flux Sampling Framework using Big Data and HPC. In *PEARC '19: Practice and Experience in Advanced Research Computing, July 22 – August 1, 2019, Chicago, IL, USA.* ACM, New York, NY, USA, 8 pages.

PEARC '19, July 22 – August 1, 2019, Chicago, IL, USA

ACM ISBN 978-1-4503-9999-9/18/06…$15.00
DOI: 10.1145/3332186.3332205

1 INTRODUCTION

The kinetics relevant to several processes in condensed matter physics such as protein folding, transport through membranes, bubble formation, and crystallization are difficult to study through straightforward molecular dynamics (MD) or Monte Carlo (MC) simulations. This is because the time between occurrences of these rare events can be much longer than the practically accessible timescales of the simulations. At typical MD simulation lengthscales of a few nanometers, observing rare events such as crystallization often requires microseconds long simulations. These simulations can take several months of computational time for molecular systems like all-atom water models. Given that several hundred rare events are necessary to obtain statistically relevant rate estimates, it is computationally prohibitive to study rare event transitions through straightforward MD (or MC) simulations.

One such process of interest in our research is crystal (e.g., ice) nucleation. Homogeneous and heterogeneous ice nucleation are relevant to atmospheric chemistry and have a significant impact on the climate and weather [19]. The kinetic details such as nucleation rates and mechanisms of ice nucleation, especially in case of heterogeneous ice nucleation, have remained elusive due to several difficulties. For example, in experimental studies the nucleation rates calculated are very sensitive to the technique used [9]. Further, the lengthscales (involving few hundreds of water molecules) and the timescales at which nucleation proceeds are hard to probe in experiments. On the other hand, molecular simulations are designed for these length- and time-scales, making them ideally suited for studying ice nucleation. However, since ice nucleation is a rare event, sampling sufficient nucleation events is challenging.

Several techniques [3, 7, 20, 21] have been developed to sample rare events in simulations and are collectively referred to as rare event methods. One such technique is forward flux sampling (FFS) [2, 3]. In FFS, simulations from initial state A to final state B are propagated through non-overlapping interfaces between A and B (see Fig. 1). This approach breaks down the low probability A-to-B transition into multiple relatively more probable transitions between intermediate interfaces. Compared with other advanced

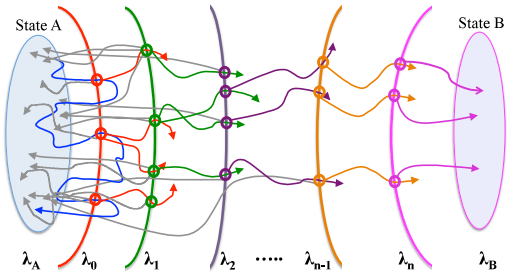

Figure 1: Conceptual overview of FFS [2]. The basin simulation is shown as the blue path. Circles represent configurations that are stored at each interface. Trajectories are shown as arrows: trajectories that cross the next interface are colored based on the interface from which they originate; trajectories that return to the basin are shown in gray. λ_i represents the ith interface between the A and B basins.

sampling methods, FFS has several advantages, including applicability to equilibrium and non-equilibrium systems and a comparatively simple and embarrassingly parallel algorithm. The challenge in implementing FFS is that as the difficulty of the problem increases, that is, the probability of the A-to-B transition decreases, FFS becomes extremely computationally demanding. Correspondingly, the amount of data and the number of tasks become difficult to handle with traditional scripting tools. We have experienced this in our studies of heterogeneous ice nucleation.

Motivated by this, we have developed a software framework called **S**calable **A**utomated **FFS** **f**or **I**lluminating **R**are **E**vents (SAFFIRE). Our framework utilizes Cascading [22] and Hadoop [23] to handle the large number of tasks and amount of data required for large scale FFS simulations.[1] In this paper we describe the details of the framework and its scalability, compare our approach to other FFS software, and discuss scientific research enabled by SAFFIRE.

2 FFS WORKFLOW

The goal of FFS is to divide the extremely low probability A-to-B transition into higher probability transitions between interfaces along the A-to-B pathway (Fig. 1). Interfaces are defined by specific values of an order parameter (λ) that can distinguish between state A and state B. For example, in ice nucleation the number of ice-like water molecules can be used as the order parameter – this value grows as the system transitions from liquid to solid.

FFS starts at the first interface, λ_0. Configurations for this interface are obtained from simulations in the initial state A (λ<λ_A), also known as the "basin". For each configuration at λ_0 (λ_i), several trial simulations are executed using a standard computational code. These simulations are analyzed to identify the next interface, λ_1 (λ_{i+1}). Each simulation is then examined to determine whether the simulation trajectory crossed λ_1 (λ_{i+1}) or returned to the basin. If the simulation crossed λ_1 (λ_{i+1}), the configuration of the system at the instant when the simulation reaches the next interface is added to the set of configurations for λ_1 (λ_{i+1}). This set of configurations is used to generate trial simulations from λ_1 (λ_{i+1}). This process is continued until the final interface, λ_n, is reached.

The flow diagram for our implementation of FFS is shown in Figure 2. There are three possible outcomes for any trial simulation: (i) the simulation crosses the basin (λ_A) before the next interface (λ_{i+1}), (ii) the simulation crosses λ_{i+1} before the λ_A or (iii) neither (i) or (ii) outcome is obtained. These are referred to as *Fallback*, *Complete*, and *Incomplete*, respectively. As the system moves away from the basin, the simulation time required for simulations to either cross the λ_{i+1} or λ_A becomes longer and longer. As a result, two categories of simulations are used. First, "short simulations" are performed, which enable the identification of λ_{i+1} and the status of the majority of the trajectories. In the case of Incomplete simulations, the simulation neither crosses λ_{i+1} nor λ_A in the allotted simulation time. This indicates that the simulation has not run long enough. Incomplete simulations are then extended with "long simulations" until they finish running (become Complete or Fallback). All Complete simulations are then analyzed to generate new configurations for the next interface. The final counts of Fallback and Complete simulations provides the probability of reaching λ_{i+1} from λ_i, $P(\lambda_{i+1}|\lambda_i)$. Once all N interfaces are complete, the product of these probabilities $\prod_{i=0}^{N-1} P(\lambda_{i+1}|\lambda_i)$ is used to estimate the rate of occurrence of the rare event – the transition from initial state A to final state B.

3 USER REQUIREMENTS

Prior to this work, a framework guiding the FFS workflow was implemented with Bash scripts. It was executed on the campus supercomputer using standard file system support and no additional data infrastructure. Due to I/O bottlenecks, this prototype executed for weeks to complete a small FFS simulation. From this implementation, we learned that for our scientific problems the majority of the simulations require very short execution times (e.g, <5 minutes). However, perhaps millions of simulations are required to successfully complete the FFS simulation. Therefore, it is important to have an infrastructure that can support high throughput computing. Secondly, each simulation produces a modest sized file. The result is that the overall application produces a massive amount of intermediate data from each interface of the FFS simulation. These files are written to the file system and there can be millions of files at any given time. On our campus supercomputing cluster consisting of separate compute and storage nodes, moving, storing, and analyzing this data is a bottleneck for the FFS simulation. In addition, the heavy load leads to instability of the parallel file system. Therefore, we needed to address both the issues of high task throughput and a large number of files.

We identified the following user requirements necessary in a software framework designed to support large scale FFS simulations:

- **Tolerance to single node failure:** The scope and scale of the target application demand substantial computing resources, and the execution times are typically measured in hours or days. Therefore, the framework should be resilient and fault-tolerant to single node and single task failures.

[1] *FFS simulation* refers to a complete execution of the FFS algorithm, whereas *simulation* refers to a molecular simulation (i.e. MD or MC simulation) which is part of the FFS algorithm.

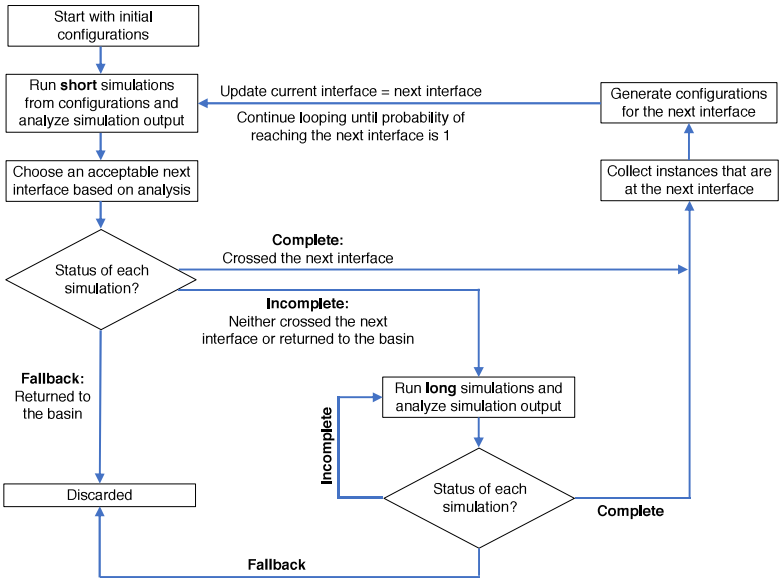

Figure 2: Flow diagram for FFS as implemented in SAFFIRE.

- **Massive data transfer:** FFS requires analysis of aggregated simulation output to determine the status of each simulation and to calculate the probability of advancing to the next interface. The size of the aggregated output requires support for massive data transfer and management capability.
- **Flexible user configuration:** The user needs freedom to choose among different simulation software and analysis tools, depending on the science problem. The framework should also offer flexibility in defining parameters for FFS, such as the number of short simulations, the minimum number of configurations necessary at each interface, etc.
- **Dynamic resource allocation:** Given that the application is being executed in a shared computing environment, it is beneficial for the framework to take advantage of additional resources, or fewer resources, at any time during the execution process in an automated, dynamic, and transparent manner.
- **Usability by a broad community:** The necessary steps for installing, configuring, running, and fine-tuning the framework should be as simple as possible. While not a technical requirement, it is very important in promoting the adoption of the framework by a broad community.

4 FFS FRAMEWORK IMPLEMENTATION

SAFFIRE is a comprehensive software framework designed to address the high throughput and data intensive computing challenges presented by FFS. SAFFIRE utilizes the Hadoop infrastructure to control the massive number of individual simulation instances and subsequent output, with the Cascading libraries [22] to manage the overall workflow.

4.1 Hadoop and Cascading

Hadoop is an open source large scale computing infrastructure that can support the management and processing of a large amount of data based on the principle of data locality [23]. The core Hadoop components include the Hadoop Distributed File System (HDFS) [8] and Hadoop MapReduce (MR) [4]. HDFS is composed of a single centralized management node called the *NameNode*, which maintains all the metadata for the Hadoop infrastructure, along with multiple storage nodes called *DataNodes*, which contain all the data in large block sizes. The MR computation model includes a single central management node called the *ResourceManager*, which is responsible for delegating the specific map and reduce tasks for a submitted MR job to a subset of the NodeManagers located on multiple computation nodes. The DataNodes and NodeManagers exist on the same physical computing system and are connected via communication between the NameNode and the ResourceManager to provide the computation and data locality integration. This is critical to the performance of large-scale data processing. The working mechanisms and performance characteristics of HDFS and MR are well studied [12]. The Hadoop infrastructure comes with features such as scalability, high fault-tolerance, and automated error recovery.

Cascading is a platform that supports the development of complex data-driven applications on the Hadoop infrastructure. Cascading accomplishes this goal by abstracting away the interaction between the developers and the data stored in HDFS. Data dependencies among the different modules or functions of a complex multi-stage application are viewed as data flows, or "pipes". These data pipes can be manipulated through operations such as filter, merge, split, and redirect. This level of abstraction allows the developer to focus more on the architectural flow of the applications in a plug-and-play manner, rather than the minute interactions with the underlying data.

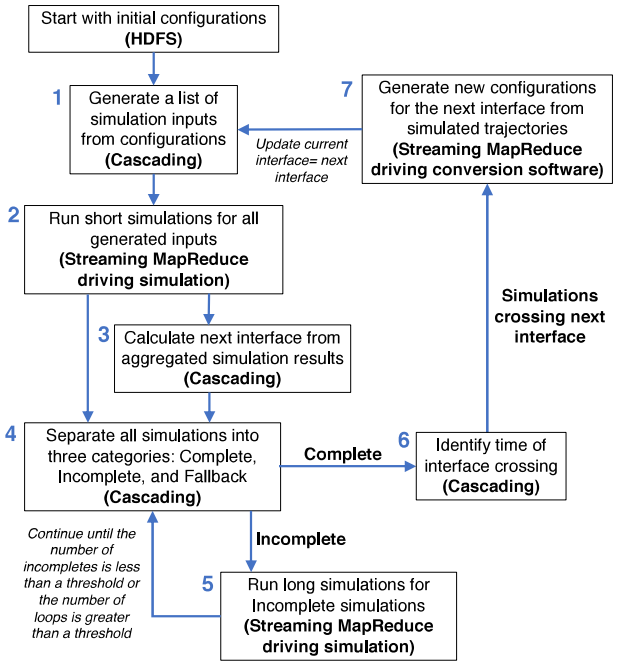

Figure 3: Architectural diagram of SAFFIRE.

4.2 Implementation Details and Features of SAFFIRE

Key components of SAFFIRE are matched to workflow steps in Figure 3. Streaming MR is shown in three components. The first streaming MapReduce (MR) job drives an external program to run the short simulations (box 2). A second streaming MR job drives an external program to run simulations to complete the Incompletes (box 5). A third streaming MR job drives conversion software to generate the set of configurations at the next interface from the simulation trajectories (box 7). Using streaming MR to control external executables enables use of a wide range of simulation engines. We have successfully tested simulation platforms such as GROMACS [18] and LAMMPS [17]. Other software packages for any type of simulation and analysis can easily be integrated with SAFFIRE with no code modifications.

Cascading acts as a flow manager and allows these steps to be executed as a single logical unit while maintaining workload dependencies among the steps. Cascading's ability to manipulate both MR modules and flows of data between these modules enables the addition of data-centric tools into SAFFIRE for the purpose of analyzing intermediate data without impacting the main FFS process. The Cascading workflow implementation is based on previous work in data flow management [24].

The combination of Hadoop and Cascading provides SAFFIRE a number of features that address the application and user requirements outlined in Section 2. Through Hadoop, SAFFIRE has the ability to manage the allocated computing resources from user space via the JobTracker's customized scheduler. Hadoop Distributed File System (HDFS) provides large scale data management infrastructure with data locality and data redundancy. The Hadoop platform has mechanisms to automatically support fault-tolerance and error recovery through data replication and job/task re-execution. This renders SAFFIRE a framework that has a high level of fault-tolerance and error recovery capability. We have additionally incorporated a simple user interface into SAFFIRE. Users can modify FFS parameters such as the number of interfaces, the number of simulations per interface, a threshold value used for interface selection, and more. The availability of dynamic Hadoop clusters similar to [11] allows SAFFIRE to be run on any traditional HPC environments. These capabilities improve the usability of SAFFIRE.

The Cascading/Hadoop-based implementation requires only user privileges. No administrative privileges are required to install and run SAFFIRE for the default deployment. This capability has been demonstrated in research and education projects using Hadoop-based environment at scale on the Clemson Palmetto computing cluster [15, 16]. This provides SAFFIRE a high degree of interoperability on different institutional and community platforms such as XSEDE.

5 SCALABILITY EVALUATION

The performance of SAFFIRE is evaluated and discussed in this section, as follows. First, we focus on the scalability of the application with respect to the number of cores and size of the problem using both strong and weak scaling. Second, the behavior of the application (e.g., computation and data transfer) is profiled and characterized under different execution scenarios. Finally, the effects of phases of application performance are characterized under different execution scenarios. Our testbed is part of Clemson University's Palmetto Supercomputer, from which we can provision isolated dynamic clusters to deploy the Cascading/Hadoop environment. Throughout the performance evaluation, the individual compute nodes provisioned for the different experimental clusters are consistently configured with 16-core Intel Xeon E5-2665 CPUs, 64GB of memory, 900GB local HDD, and 300GB local SSD. The system used for the performance analysis was the early stages of homogeneous ice nucleation in the mW water model [14] at 230 K and 1 atm. The system comprised of 4096 water molecules and each simulation was executed for 3 ps (time step = 0.002 ps) of molecular dynamics in LAMMPS (`https://lammps.sandia.gov`) [17]. The order parameter used to quantify the progress of each simulation, λ, was the size of the largest cluster of ice-like water molecules defined with the procedure from Ref. 13

5.1 Scalability Analysis

Both strong and weak scaling are considered in the scalability analysis of SAFFIRE. For strong scaling analysis, the problem size (number of simulations per interface[2]) is held constant and the number of cores in the Hadoop cluster used to run SAFFIRE is increased. For weak scaling analysis, the problem size is increased in equal proportion to the increase in the number of cores, so that the amount of work per core remains constant. For both strong and weak scaling analysis, we consider the performance of SAFFIRE for four interfaces of FFS with 10,000 simulations per interface on a 128

[2]For the remainder of the text we use 'number of simulations' in place of 'number of simulations per interface' for brevity (i.e. a 128 core cluster with 10,000 simulations, has 10,000 simulations *per interface*).

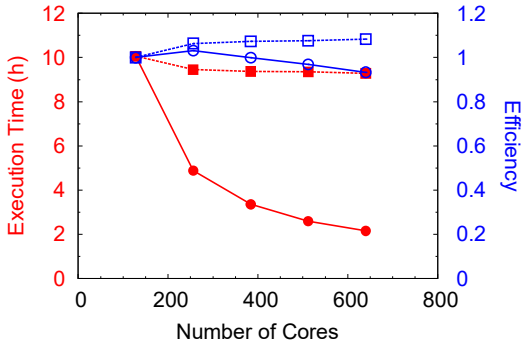

Figure 4: Strong and weak scaling of SAFFIRE. Execution time (red, filled markers) is read from the left axis, and scaling efficiency (blue, open markers) from right axis. Strong scaling is shown with circles lines, and weak scaling is shown with squares lines. All values are reported at 95% confidence.

core Hadoop cluster as the baseline performance. Strong scaling efficiency, E_{strong} is calculated as:

$$E_{strong} = \frac{t_{128}}{N t_n} \tag{1}$$

where t_{128} is the execution time for SAFFIRE on 128 cores (8 nodes), t_n is the execution time for the application on n cores, and N is number of cores in the Hadoop cluster divided by the baseline 128 cores ($n/128$). Weak scaling efficiency, E_{weak} is calculated as:

$$E_{weak} = \frac{t_{128}}{t'_n} \tag{2}$$

where, t_{128} is the execution time for the application on 128 cores, and t'_n is the execution time for a problem size N times larger than the baseline problem, executed on n cores, where $n = 128 \times N$.

The scaling performance of SAFFIRE is shown in Figure 4 where each test consisted of running four FFS interfaces. Since each FFS interface comprises a similar operation, our results are not expected to change with a larger number of interfaces. The application displays excellent strong scaling performance to more than 600 cores (40 nodes). The execution time drops from nearly 10 hours when running on a Hadoop cluster with 128 cores to just over 2 hours on a 640 core Hadoop cluster. The strong scaling efficiency remains over 90% for all systems tested, however the strong scaling efficiency generally decreases as more cores are added. This may be due to the increased overhead of a larger Hadoop cluster, and increased data transfer times to copy simulation data from HDFS to local scratch and simulation results from local scratch to HDFS. These data transfers are initiated from within the Hadoop Streaming map tasks, and therefore are unable to take advantage of the built-in data-locality offered by Hadoop. As such, when the Hadoop cluster increases in size, the data must be transferred further across the network. Additionally, since HDFS is spread across an increased number of nodes, the likelihood of finding the necessary simulation data already on the node performing the map task decreases. More discussion of data transfer overhead follows later in this section. Larger Hadoop clusters also have the possibility of an increased number of idle nodes if the number of tasks is not divisible by

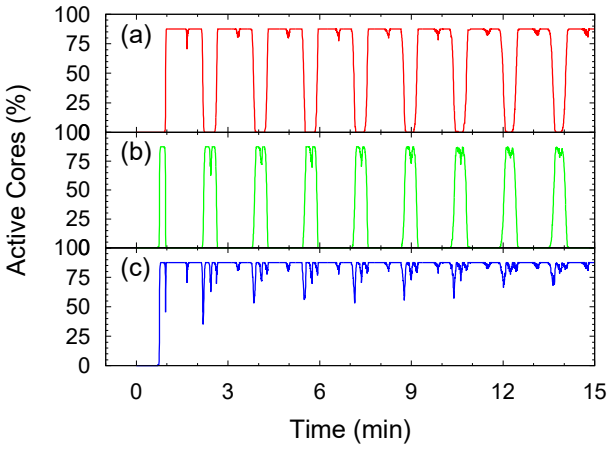

Figure 5: Snapshot of application profiling over a 15 minute interval. Percent of cores in the cluster which are (a) performing a computation (simulation or analysis), (b) performing a data transfer, or (c) either performing a computation or data transfer at a given time.

the number of cores available to execute the tasks. We term these "remainder effects" and explore them in detail later in this section.

SAFFIRE weak scaling performance is also shown in Figure 4. The overall execution time of the application decreases as the problem size is increased in proportion to the number of cores in the Hadoop cluster. The weak scaling efficiency increases over the range of Hadoop cluster sizes studied, resulting in a weak scaling efficiency that is always greater than or equal to 1. One source for the increase in efficiency is related to how we chose to scale the size of the job in the weak scaling analysis. In our implementation, the head node of the Hadoop cluster does not perform any computation. Therefore, when the number of total cores in the cluster increases, only the number of slave node cores increases – meaning the computational resources available for task execution grows faster than the problem size. While it is possible to take advantage of unused cores on the head node for computational purposes, the amount of memory held by the NameNode and the ResourceManager processes to maintain metadata for the massive amount of data, file counts, and map/reduce tasks makes it impractical to do so. It is also possible to use the number of slave node cores rather than total cores when calculating efficiency. However, we use total cores in the calculation because the head node resources are required to manage the Hadoop cluster, even if they are not being used for scientific computation. The cluster (and therefore application) management overhead decreases in terms of the percentage of total cores with increasing cluster size, and this manifests itself by contributing to the weak scaling performance of the application. Together the strong and weak scaling results highlight the scalability of SAFFIRE.

5.2 Application Profiling

An analysis of application behavior was performed. The primary goal of SAFFIRE is to efficiently enable the execution of a large number of simulations for FFS. A core is performing useful work when it is running a simulation, analyzing simulation output, or

performing a necessary file conversion. All other time is considered application overhead. Realistically speaking, SAFFIRE also manages the simulation output and automates the FFS algorithm. Though these tasks save the end-user time and effort, they are not computation-intensive tasks as compared with the execution and analysis of the simulations. Therefore, as a starting point to evaluate application behavior and framework overhead, we focus on profiling the Hadoop Streaming tasks that are responsible for running the simulations and analysis.

Each Hadoop Streaming map task is executed on one core. Logging capabilities were added to the Hadoop Streaming tasks using the $gettimeofday()$ function to record the start and stop times for each simulation, analysis, file conversion, and data transfer. The events were aggregated to create a representation of the number of cores active with each task type across the entire Hadoop cluster at any given instance in time. Logging the start and stop times did not significantly change the overall execution time of SAFFIRE within the 95% confidence interval.

The different types of core activity were grouped into computation (e.g., simulation, analysis, and file conversion), and data transfers (e.g., file transfers between local scratch and HDFS, which are initiated from within the Hadoop Streaming map tasks). Figure 5(a) shows the percentage of all cores performing a computation activity within the 15 minute snapshot. Initially, no cores are active with computation until the ~1 minute mark, when the first Hadoop Streaming tasks begin. Nearly simultaneously, all the cores on slave nodes are consumed with computation. Note that the percentage of active cores reaches a maximum of about 85% because we report the percentage of active cores with reference to the total size of the Hadoop cluster, not just the number of slave node cores. The 128 core Hadoop cluster shown in Figure 5 has up to 112 active cores at one time with the remaining non-active 16 cores of the head node. Evidence of the small time gap between the simulation and analysis appears as a brief decrease in the percentage of active cores between the one and two minute marks. Just past two minutes, none of the cores are involved in computation. In Figure 5(b) the percentage of cores involved in data transfers is shown. The cores are involved in a data transfer just before the first computation (Figure 5(a)), because the Hadoop Streaming task must copy a configuration file from HDFS to local scratch to initiate the simulation. After the first batch of computation, another batch of data transfers appears as the Hadoop Streaming tasks copy simulation output from local scratch to HDFS. A brief decrease in the data transfer appears (e.g. 2.5 minutes), marking the distinction between the data upload to HDFS from the first batch of simulations, and the data download to local scratch for the second batch of simulations. In Figure 5(c), the computation and data activity are combined to report an overall percent of active cores over time.

Several interesting features of SAFFIRE behavior are apparent from Figure 5. The simulations are executed across the entire cluster in a batch manner, with all slave node cores actively performing computations and data transfers at nearly the same time. This job submission pattern holds through several batches of Hadoop Streaming map tasks. From the small dips in overall core activity in Figure 5(c) there is limited aggregate core downtime between each batch of Hadoop Streaming tasks. The data transfers between local scratch and HDFS also contribute noticeable overhead to SAFFIRE.

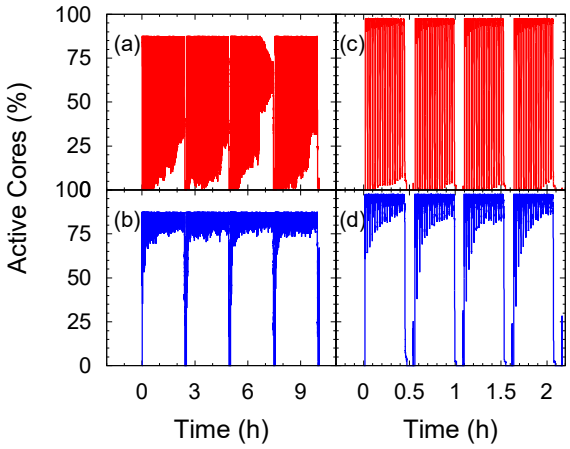

Figure 6: Application profiling: Percent of cores in the 128 core cluster with 10,000 simulations which are (a) performing a computation (e.g., simulation, analysis and file conversion), and (b) either performing a computation or data transfer at a given time. Percent of cores in the 640 core cluster with 10,000 simulations which are (c) performing a computation, and (d) either performing a computation or data transfer at a given time.

If a SAFFIRE user had a system that required large file transfers to run a short (in wall clock time) simulation, the data transfer overhead would detrimentally impact the performance of SAFFIRE.

Figure 6(a)-(b) shows application profiling across four complete interfaces (i.e. λ_i to λ_{i+4}) of a FFS simulation for the same system profiled in Figure 5. For comparison, data is also shown for the case with 640 cores and 10,000 simulations in Figure 6(c)-(d). The computation for each FFS interface can be identified as the groupings of core activity (the first of which is from 0 to 2.5 hours and 0 to 0.5 hours for Figure 6(a)-(b) and 6(c)-(d), respectively), and they are separated by the time interval where the cores are performing neither computation nor data transfers. In these regions, the Cascading code is parsing though the results of the analysis and picking the next interface. Some amount of computation is also performed to generate the new configurations, however it is too short to appear in the plots. The data transfer required for the file conversion appears as the short vertical line in advance of the larger batch of core activity for interfaces 2, 3, and 4 in Figure 6(d). The same feature exists in Figure 6(b), but is not visible due to the scale of the figure. In Figure 6(c)-(d), the batch-like Hadoop Streaming task execution is maintained across each interface. Near the end of the first and last interfaces, the task execution appears to be less synchronized as evidenced by the lines in Figure 6(c) not quite reaching zero activity at that part of the execution.

From Figure 6(a), it can be seen that the Hadoop Streaming job submission has a "synchronized" nature for the first ~30 minutes for each interface for the system with 128 cores and 10,000 simulations. After about 30 minutes, the Hadoop Streaming jobs do not appear to be synchronized. Some cores are performing computation, while others are performing a data transfer. Neither the white space that appears below the red lines in Figure 6(a) as each interface progresses nor above the red lines as interface 3 progresses

indicate that the cores are less active overall – just that the Hadoop Streaming task execution does not follow a synchronized pattern after about 30 minutes.

5.3 Remainder Effects

The appearance of synchronization of the Hadoop Streaming task execution led us to investigate whether making the number of simulations a multiple of the number of slave node cores would decrease the execution time by eliminating "remainder effects", where some cores in the Hadoop cluster are idle while the last partial batch of Hadoop Streaming tasks are completed. Based on the application profiling, we expect that the remainder effects will be most prominent for the system with 640 cores and 10,000 simulations. We also tested the system with 128 cores and 10,000 simulations for comparison. The remainder effects are tested using FFS simulations in which the number of simulations is a multiple of the number of slave node cores (9984 simulations for 624 slave cores, 10080 simulations for 112 slave cores) and then tested using another run in which one additional simulation is added (9985 simulations for 624 slave cores, 10081 simulations for 112 slave cores). This setup tests a worst-case scenario. If the Hadoop Streaming jobs display perfect batch behavior then in the worst case scenario all except one slave node core will be idle when the last simulation is completing. Each test was performed in triplicates to calculate the 95% confidence interval of our results.

Visual inspection of the application profiles (not shown) does not reveal any clear differences between the perfect match and worst case scenario application runs. However, as reported in Table 1, there are differences in the execution times. As expected, the remainder effects are the most prominent for the simulation that has the most synchronized-like Hadoop Streaming task execution. For a Hadoop cluster size of 128 cores (112 slave cores), the remainder effects have no significant effect on the execution time. Based on the application profiling seen above (Figure 6), this is not surprising because for this setup the synchronous Hadoop Streaming task submission pattern is not present at the end of an interface. For a Hadoop cluster with 640 cores (624 slave cores), the execution time for the system with no remainder simulations is about 200 seconds faster than the worst case scenario. Although this demonstrates that remainder effects can impact the execution time, they represent a small fraction of the overall execution time.

Table 1: Execution time and percent of time that the cores were active for systems with possible remainder effects.

Cores	128	128
Simulations	10080	10081
Execution Time (s)	36390 ± 68	36453 ± 140
Computation (%)	62.0 ± 0.2	61.9 ± 0.2
Any Activity (%)	81.8 ± 0.2	81.7 ± 0.1
Cores	640	640
Simulations	9984	9985
Execution Time (s)	7570 ± 73	7758 ± 30
Computation (%)	59.2 ± 0.6	57.7 ± 0.3
Any Activity (%)	78.2 ± 0.8	76.3 ± 0.3

6 RELATED WORK

To our knowledge, there are only two other significant efforts in implementing FFS-based simulation techniques, the Flexible Rare Event Sampling Harness System (FRESHS) [10] and Parallel Forward Flux Sampling (PFFS) [1]. The FRESHS architecture includes a FRESHS server and multiple FRESHS clients, all implemented in Python. The server is responsible for accepting parameters for the rare-event simulation, asynchronous communications from clients, and an SQLite database to store data for intermediate interfaces. The server tracks the progress of FFS, while clients are responsible for the simulations and analysis (order parameter calculation).

Similar to SAFFIRE, the goal of FRESHS is to provide a parallelized FFS implementation that allows users to insert various simulation softwares. Beyond the architectural differences, SAFFIRE and FRESHS also differ in the implementation of the FFS algorithm. FRESHS uses the exploring scouts technique [10]. This technique works well when the analysis is run from within the simulation program – however this often requires modification of the simulation software source code. Though FRESHS can be used with separate simulation and analysis codes, it requires extremely short simulations that incur large startup and shutdown overhead. SAFFIRE is specifically designed for separate simulation and analysis codes for maximum user flexibility. The SAFFIRE and FRESHS implementations both have advantages and disadvantages depending on the system and simulation software. Unfortunately their differences make a meaningful performance comparison difficult.

PFFS implements FFS using the C programming language and MPI to support parallelism [1]. In the original design, PFFS is implemented as a single large-scale FFS simulation program. PFFS requires researchers to recompile from source to include custom simulation engines. It also uses individual files on the shared file system to store simulation results which can become difficult to scale as the number of simulations in FFS reaches millions or even billions. The nature of MPI is also a disadvantage of the PFFS implementation as compared to the Hadoop implementation of SAFFIRE. MPI is not typically tolerant to single node or task failures in the parallel computing environment, whereas Hadoop provides fault-tolerance in the case of single node failures. PFFS never matures out of the testing stages, and no software package is available for testing purposes.

7 SAFFIRE ENABLED SCIENCE

Our research group is actively using SAFFIRE to study several scientific problems. In addition to the case of heterogeneous ice nucleation described in the introduction, we have also used SAFFIRE to study the nucleation of clathrate hydrates [5] and Lennard–Jones particles. Clathrate hydrates are a crystal composed of water and guest (e.g., methane) molecules. Their formation presents substantial safety hazards in oil and gas transportation. In addition, researchers are exploring hydrates for technological applications in natural gas storage and gas separations. To investigate the mechanism of hydrate nucleation, we used molecular dynamics simulations with FFS comprising 10 interfaces and 10,000–40,000 short simulations per interface. Including the long simulations, SAFFIRE managed over 500,000 individual simulations. The entire calculation required 33 days on a 30-node Hadoop cluster with the same

per-node specifications as listed in the scalability evaluation. From the ~1000 nucleation pathways generated by SAFFIRE, we performed one of the most comprehensive evaluations of the hydrate nucleation mechanism to date [5]. We are currently using SAFFIRE to study the effects of guest solubility on the hydrate nucleation mechanism.

Our group also actively develops new methods to study rare events, and recently published a method called contour FFS (cFFS) [6]. cFFS allows FFS to be performed along multiple order parameters simultaneously. This improves the effectiveness FFS in cases where there are multiple transition tubes and/or the optimal order parameter is not known *a priori*. We are planning to implement cFFS in SAFFIRE.

8 CONCLUSION

This paper describes the design, implementation, and performance evaluation of SAFFIRE, a data-intensive computing platform that implements the FFS technique for rare events in simulations. The computational and data intensive demands of FFS are supported through the Cascading/Hadoop-based implementation with features such as implicit parallelism, user-level management of allocated resources, robust infrastructure for large-scale data movement and management, and ease of user access. Performance characterization and evaluation demonstrates the robustness and scalability of SAFFIRE. When the number of simulations is kept constant and the number of processing cores is increased (strong scaling), SAFFIRE scaling efficiency remains over 90%. We also observed scaling efficiencies greater than 1 when we increased the number of cores in proportion with the number of simulations (weak scaling). The availability of SAFFIRE has led to a number of scientific discoveries and new rare-event simulation techniques.

Our ongoing work with SAFFIRE focuses on reducing the data transfer overhead. We will investigate a number of approaches including implementing a map-only simulation server and integrating a large scale NoSQL database (e.g., HBase) on top of HDFS to handle the management and storage of simulation outcomes. As SAFFIRE matures, efforts are being made to release the software to the research community. Preliminary work has been done to generalize the configuration settings of the Hadoop infrastructure and the customization of the scientific workflows on XSEDE's Bridges. Additional work will be carried out to further decouple the setup of Hadoop (for computing sites that do not have a local Hadoop library) from SAFFIRE's workflow drivers. Detailed documentation to help with SAFFIRE's deployment as well as development of new scientific workflow is being created. We anticipate a full public release of the software later this year. In the meantime, access to SAFFIRE will be provided upon request.

ACKNOWLEDGMENTS

RSD and SS acknowledge the support by the U.S. Department of Energy, Office of Science, Office of Basic Energy Sciences, under Award No. DE-SC0015448. AA and LBN acknowledge the support by the National Science Foundation under Award No. 1405767. Simulations were performed on the Palmetto Supercomputer Cluster of Clemson University and the Bridges Supercomputer through XSEDE.

REFERENCES

[1] Rosalind Allen, Juho Lintuvori, and Kevin Stafford. 2013. Parallel Forward Flux Sampling. http://www2.epcc.ed.ac.uk/~kevin/ffs/index.html
[2] Rosalind J. Allen, Dann Frenkel, and Pieter Rein ten Wolde. 2006. Simulating rare events in equilibrium or nonequilibrium stochastic systems. *J. Chem. Phys.* 124 (2006), 024102.
[3] R. J. Allen, C. Valeriani, and P. R. ten Wolde. 2009. Forward flux sampling for rare event simulations. *J. Phys. Condens. Mat.* 21, 46 (2009).
[4] Jeffrey Dean and Sanjay Ghemawat. 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51, 1 (Jan. 2008), 107–113. https://doi.org/10.1145/1327452.1327492
[5] R. S. DeFever and S. Sarupria. 2017. Nucleation mechanism of clathrate hydrates of water-soluble guest molecules. *J. Chem. Phys.* 147, 20 (2017), 204503.
[6] R. S. DeFever and S. Sarupria. 2019. Contour forward flux sampling: Sampling rare events along multiple collective variables. *J. Chem. Phys.* 150, 2 (2019), 024103.
[7] C. Dellago, P. G. Bolhuis, and P. L. Geissler. 2002. Transition path sampling. *Adv. Chem. Phys.* 123 (2002), 1–78.
[8] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. *ACM SIGOPS operating systems review* 37, 5 (2003), 29–43.
[9] C. Hoose and O. Mohler. 2012. Heterogeneous ice nucleation on atmospheric aerosols: a review of results from laboratory experiments. *Atmos. Chem. Phys.* 12 (2012), 9817–9854.
[10] K. Kratzer, J. T. Berryman, A. Taudt, J. Zeman, and A. Arnold. 2014. The flexible rare event sampling harness system (FRESHS). *Comput. Phys. Commun.* 185, 7 (2014), 1875–1885.
[11] Sriram Krishnan, Mahidhar Tatineni, and Chaitanya Baru. 2011. myHadoop-Hadoop-on-Demand on traditional HPC resources. *San Diego Supercomputer Center Technical Report TR-2011-2, University of California, San Diego* (2011).
[12] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. 2012. Parallel data processing with MapReduce: a survey. *AcM sIGMoD Record* 40, 4 (2012), 11–20.
[13] T. Li, D. Donadio, G. Russo, and G. Galli. 2011. Homogeneous ice nucleation from supercooled water. *Phys. Chem. Chem. Phys.* 13, 44 (2011), 19807–19813.
[14] V. Molinero and E. B Moore. 2008. Water modeled as an intermediate element between carbon and silicon. *J. Phys. Chem. B* 113, 13 (2008), 4008–4016.
[15] W. C. Moody, L. B. Ngo, E. Duffy, and A. Apon. 2013. JUMMP: Job Uninterrupted Maneuverable MapReduce Platform. In *Proceedings of the 2013 IEEE International Conference on Cluster Computing*.
[16] L. B. Ngo, E. Duffy, and A. Apon. 2014. Teaching HDFS/MapReduce systems concepts to undergraduates. In *Proceedings of the NSF/TCPP Workshop on Parallel and Distributed Computing Education*.
[17] S. Plimpton. 1995. Fast parallel algorithms for short-range molecular dynamics. *J. Comput. Phys.* 117, 1 (1995), 1–19.
[18] S. Pronk, S. Pall, R. Schulz, P. Larsson, P. Bjelkmar, M. R. Apostolov, J. C. Smith, P. M. Kasson, D. van der Spoel, B. Hess, and E. Lindahl. 2013. GROMACS 4.5: a high-throughput and highly parallel open source molecular simulation toolkit. *Bioinformatics* 29 (2013), 845–854.
[19] H. R. Pruppacher and J. D. Klett. 1997. *Microphysics of Clouds and Precipitation* (2 ed.). Kluwer Academic Publishers, Dordrecht, The Netherlands.
[20] Mary A Rohrdanz, Wenwei Zheng, and Cecilia Clementi. 2013. Discovering Mountain Passes via Torchlight: Methods for the Definition of Reaction Coordinates and Pathways in Complex Macromolecular Reactions. *Ann. Rev. Phys. Chem.* 64 (2013), 295–316.
[21] T. S. Van Erp and P. G. Bolhuis. 2005. Elaborating transition interface sampling methods. *J. Comput. Phys.* 205, 1 (2005), 157–181.
[22] CK Wensel. 2015. Cascading: Defining and executing complex and fault tolerant data processing workflows on a Hadoop cluster. http://www.cascading.org
[23] Tom White. 2012. *Hadoop: The definitive guide.* O'Reilly.
[24] Pengfei X., Yueli Z., Sapna S., and Amy A. 2013. SciFlow: A dataflow-driven model architecture for scientific computing using Hadoop. In *Proceedings of the IEEE International Conference on Big Data*.