Accelerating Data-Intensive Seismic Research Through Parallel Workflow Optimization and Federated Cyberinfrastructure

Marcus Adair
Ivan Rodero
Manish Parashar
University of Utah
Scientific Computing and Imaging (SCI) Institute
Salt Lake City, UT, USA
{u1131818,ivan.rodero,manish.parashar}@utah.edu

Diego Melgar
University of Oregon
Dept. of Earth Sciences
Eugene, OR, USA
dmelgarm@uoregon.edu

ABSTRACT

Earthquake early warning systems use synthetic data from simulation frameworks like MudPy to train models for predicting the magnitudes of large earthquakes. MudPy, although powerful, has limitations: a lengthy simulation time to generate the required data, lack of user-friendliness, and no platform for discovering and sharing its data. We introduce FakeQuakes DAGMan Workflow (FDW), which utilizes Open Science Grid (OSG) for parallel computations to accelerate and streamline MudPy simulations. FDW significantly reduces runtime and increases throughput compared to a single-machine setup. Using FDW, we also explore partitioned parallel HTCondor DAGMan workflows to enhance OSG efficiency. Additionally, we investigate leveraging cyberinfrastructure, such as Virtual Data Collaboratory (VDC), for enhancing MudPy and OSG. Specifically, we simulate using Cloud bursting policies to enforce FDW job-offloading to VDC during OSG peak demand, addressing shared resource issues and user goals; we also discuss VDC's value in facilitating a platform for broad access to MudPy products.

CCS CONCEPTS

• **Computing methodologies** → *Parallel computing methodologies*; • **Human-centered computing** → *Accessibility systems and tools*; • **Applied computing** → *Earth and atmospheric sciences.*

KEYWORDS

parallel workflow, HTC, OSG, cyberinfrastructure, data democratization, earthquake early warning, VDC

ACM Reference Format:
Marcus Adair, Ivan Rodero, Manish Parashar, and Diego Melgar. 2023. Accelerating Data-Intensive Seismic Research Through Parallel Workflow Optimization and Federated Cyberinfrastructure. In *Workshops of The International Conference on High Performance Computing, Network, Storage, and Analysis (SC-W 2023), November 12–17, 2023, Denver, CO, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3624062.3624276

1 INTRODUCTION

Synthetic data from simulated large earthquakes (Mw 7.5+) have proven valuable in training artificial intelligence (AI)-based earthquake early warning (EEW) models to identify large earthquake magnitudes [14]. EEW systems provide advanced warning of dangerous events before ground motion is experienced to mitigate the profound risks posed [6, 10]. As large earthquakes are rare (e.g., in a given year, only about 15 Mw 7+ events and one Mw 8+ event occur [30]), synthetic data from simulation frameworks, such as MudPy (openly available on GitHub [17]), are relied upon for training EEW models [14]. MudPy, although powerful, has limitations: (1) running simulations to generate the required data can be time-consuming, potentially taking up to several days); (2) the software is not user-friendly; and (3) currently no platform is available for easy discovery and widespread sharing of the data generated. These limitations can hinder the broad adoption of MudPy, especially for nonprogrammers.

In this paper, we leverage existing cyberinfrastructure (CI), and specifically, the Open Science Grid (OSG), to create a high-throughput workflow tool for accelerating MudPy earthquake simulations, overcoming the framework's limitations.

Utilizing OSG's high-throughput computing (HTC) and storage capabilities [1, 24, 27], our workflow automates and streamlines the parallel execution of earthquake simulation and accelerates synthetic data generation, addressing the performance limitations of MudPy's native sequential simulations.

We specifically present the FakeQuakes DAGMan Workflow (FDW), our solution for accelerating and streamlining earthquake simulations using National Science Foundation (NSF) CI. Our experimental results demonstrate a 56.8% decrease in runtime when simulating 1,024 earthquakes in Chile using parallel computation on OSG versus on a single machine. The throughput also increases by approximately five times when running 50,000 simulations compared to 1,024 with the FDW. Using FDW, we also explore the optimal execution of HTCondor "DAGMan workflows" (which we refer to as DAGMan(s)) [13] for this use case on the OSG, specifically considering the behavior of partitioned parallel workflows. Our goal is to enhance efficiency and provide insights that can assist in meeting user-defined objectives (e.g., throughput).

Additionally, we investigate the acceleration of FDW MudPy simulations and OSG workflows more generally by developing bursting policies for OSG job-offloading to existing CI, such as the Virtual Data Collaboratory (VDC) [23]. VDC is an NSF-funded federated data CI that supports interdisciplinary and collaborative research and enables intense, data-driven science and engineering discoveries by offering efficient access to data, data services (e.g., metadata curation and data discovery), and computing capabilities to scientists through a web interface and API. We design and implement a Python-based Cloud bursting simulator to investigate how OSG job execution can be supplemented to meet user-driven goals, such as increased or constant throughput, demonstrating the potential for enhancing FDW performance by offloading jobs to VDC (which we denote as *VDC bursting*) when the OSG's resources are limited. We develop and test three OSG-tailored bursting policies that respond to congested queues, job submission gaps, and low throughput. We also discuss integrating FDW into VDC, highlighting the value in enabling a collaborative platform for seamlessly executing accelerated MudPy simulations and efficiently accessing the framework's AI-ready data products.

The main contributions of this paper are (1) using parallel computing to accelerate HTC earthquake simulations, (2) providing policies for offloading jobs to alternative environments to explore the potential of optimizing OSG workflows, and (3) exploring how workflows, such as FDW, executed on federated CI can enhance timely and equitable access to their data products and data services (i.e., MudPy). Complementary engineering contributions include building mechanisms for optimizing parallel job execution on OSG.

The rest of the paper is organized as follows: Section 2 provides an overview of work related to our research. Section 3 describes the implementation of the FDW and simulated OSG job-offloading. Section 4 describes our experimental methodology followed by the experimental results in Section 5. In Section 6 we assess the results, elaborate on integration with the VDC, and discuss limitations. Finally, Section 7 concludes the paper and outlines future work.

2 RELATED WORK

MudPy is a Python-based software package that can simulate earthquakes and other geoscience phenomena. Synthetic high-rate Global Navigation Satellite Systems (GNSS) waveforms (representing ground displacement information derived from satellites) produced by its FakeQuakes (FQs) module offer valuable training data for EEW models [14, 18–20, 26]. GNSS waveforms are essential in EEW for accurately characterizing large earthquakes [26]. Literature on natural hazards, AI-based earthquake modeling, and EEW systems is extensive and growing, with recent years witnessing significant research that leverages the MudPy framework. For example, Lin et al. [14] recently used FQs waveforms to help train an EEW algorithm that outperformed similar approaches in predicting the magnitudes of five actual, large earthquakes in Chile. Successful EEW deep learning algorithms prior to Lin et al. did not focus on large earthquake events, which are the most dangerous.

The products of the FQs module have been validated against actual earthquake events. Goldberg and Melgar [11] conducted earthquake simulations based on a 2014 Mw 8.1 Chilean earthquake (using the same geometry data as described in Section 4), and then compared the synthetic FQs data with the authentic earthquake observations. Their analysis confirms the reliability of FQs in simulating large events by demonstrating good agreement with observed data in both frequency and time domains (see Fig. 1 for visualization of FQs' products).

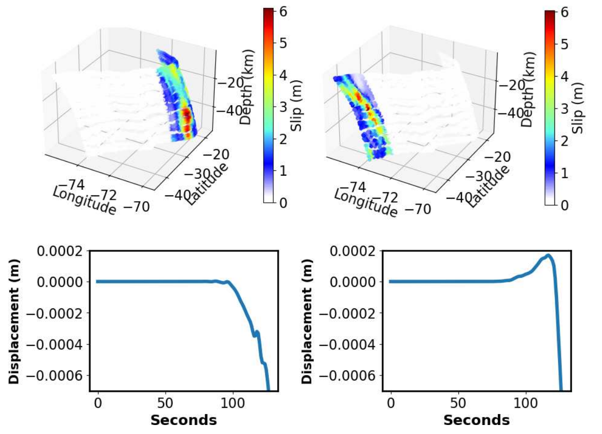

Figure 1: Simulation visualizations. These graphs depict examples of simulated ruptures (top) and GNSS waveforms (bottom) from the FDW.

A recent study enhancing FQs performance conducted by Carillo [8] proposed modifications to MudPy's code and the use of GPU-equipped machines to speed up the software. He achieved a 10% improvement using this strategy, which was deemed not feasible at scale. Carillo suggested using HTC to improve performance, which motivates the parallelization strategies in this paper. Notably, MudPy already incorporates MPI and has some parallelism, but the FQs module can be further parallelized due to its stochastic nature [16], enabling concurrent simulation of earthquakes. Despite the existing gap in research involving HTC and MudPy, various studies utilize FQs. Notably, most existing work utilizing MudPy has focused on geosciences and natural hazards [14, 18, 20, 26]. We see untapped potential in the application and see enhancing its capabilities as an opportunity to foster further research.

Cloud bursting, a technique that extends enterprise resource functionalities by leveraging public Cloud capabilities [15], has been explored in prior research. Sfiligoi et al. [28, 29] utilized Cloud bursting across three major Cloud platforms (e.g., Amazon) in support of photon propagation simulation at the IceCube Neutrino Observatory, an NSF major facility in Antarctica. IceCube employs HTCondor as a workflow manager, partially executing workloads on local on-prem infrastructure and partially on OSG. Their successful results demonstrate runtime enhancements when applying Cloud bursting to workflows partially using OSG. Although our work does not delve into inventing new Cloud bursting techniques, we contribute OSG-tailored bursting policies designed for workflows exclusively utilizing OSG.

To our knowledge, work related to partitioning approaches and configurations for DAGMans on the OSG has yet to be published. This paper describes our experiences and insights regarding DAGMan behaviors through FDW to help researchers use OSG most efficiently when executing workloads.

3 PARALLEL WORKFLOW IMPLEMENTATION

To enhance FQs speed and user experience, we leveraged the OSG's distributed processing and storage capabilities via its Open Science Pool (OSPool). In this paper, we use the terms OSG and OSPool interchangeably. The OSG facilitates the automated execution of custom software on shared computing resources contributed by research collaborators such as universities and government-supported supercomputing institutions. FQs simulations align well with the OSG, meeting "ideal" or "still advantageous" criteria for OSPool job specifications [21]. We ran thousands of concurrent jobs, each with a wall time of under 10 hours, using *4 CPU cores*, which is ideal (running jobs with *8 CPU cores* or more would not be ideal). If we instead ran hundreds of jobs or less, for example, each taking over 10 hours, this would not be ideal on OSG. Each job's input data in the FDW was less than *10GB*, which is still advantageous according to documentation (less than *500MB* would be ideal).

The OSG integrates with HTCondor, a framework for automating and managing high-throughput workflows [4]. We employed HTCondor's "DAGMan workflows" tool [13] to parallelize and automate the steps of FQs. We can integrate with other workflow engines, such as the powerful Pegasus [9], but we initially chose HTCondor's default, simpler engine DAGMan to assess the suitability of parallelizing FQs. HTCondor uses "submit description files" to specify job compute requirements, orchestrate scripts on OSG nodes, and handle input files. Parallel FDW jobs on nodes across the OSPool use *4 CPU cores* and dynamically request varying amounts of *disk* and *memory*, up to *16GB* (depending on if jobs need to generate large matrix files). If needed, our workflow tool could be launched via the VDC portal's graphical user interface (GUI). Presently, it can be run directly on the OSG by placing the source code in a OSG home directory; editing a configuration file for simulation parameters; placing input files in a single, specified directory; and running a script. The FDW's streamlined process eliminates the need to manually install packages, edit complex files, move files between steps, and more when executing FQs simulations.

MudPy was installed in a Singularity image (now Apptainer) [2] with a custom Conda environment [3] since Python and other dependencies are necessary for its operation, which all jobs running across the OSPool utilize. To facilitate faster delivery of large files to and from execute nodes, the OSG employs caching tools. We utilize their Stash Cache (now OSDF Cache [22]) to distribute the *928MB* Singularity image across the OSPool efficiently. We have developed a system to monitor the progress of running and completed DAGMans to improve MudPy with statistics on running and postprocessing times for generated synthetic data: Shell scripts parse HTCondor log files to extract information (e.g., runtime, wait times, and complete/failed job count) and compute job states and durations, enhancing simulation analysis and enabling policy implementation. After simulation, thousands of files are congregated, labeled, and archived on OSG storage capacity.

3.0.1 DAGMan Workflow Phases. The FDW consists of three phases, as described in the listing below. The phases run sequentially, with the numerous jobs of each one executed in parallel. Each phase utilizes a distinct script that executes on OSPool nodes to establish the required, "rigid" [16] MudPy folder structure, perform distinct FQs steps, and compress the output.

- **A Phase** simulates rupture scenarios in parallel. To do so, MudPy requires two recyclable "distance matrix" files (*.npy*); generating these files is time-consuming, so recycling them is crucial. In this phase, if no *.npy* files are provided, a single job will create the matrices, which parallel jobs will then use in this phase.
- **B Phase** generates Green's functions (GF) matrices, required by the next phase, as *.mseed* files. This process can span multiple hours depending on the length of a required input list of GNSS stations.
- **C Phase** consists of simulating the requested number of waveforms (based on the ruptures) in parallel to produce the final, desired output. To help expedite the delivery time of the large, compressed *.mseed* files (possibly exceeding *1GB*) to OSG nodes in this phase, we use Stash Cache (which is the same for *.npy* files).

3.1 VDC Bursting Simulator Implementation

After implementing the FDW, we constructed a VDC bursting simulation framework in Python. Instead of actual Cloud job execution, we mimicked execution times and associated costs. The baseline times used by the bursting simulator for completing offloaded jobs (see 3.1.1) were derived from statistics using a single Amazon AWS equipped with *4 Intel(R) Xeon(R) Platinum 8175M CPUs* and the Singularity image from Section 3 to automatically run MudPy. Using the AWS Cloud Machine, we generated the same quantity of synthetic MudPy data as individual OSG jobs do while employing identical FQs parameters as in Section 4 to calculate average Cloud job times used by the simulator. Amazon AWS was chosen due to VDC's capability of connecting Cloud services, such as Amazon [23]. In practice, bursting will target physical and Cloud nodes managed by VDC.

This bursting simulator requires two *.csv* files as input that contain the submission, execution, and termination times of an actual DAGMan batch and the same information for individual jobs within it. These times are used to iterate through the batch's runtime, monitor job status, and simulate the potential performance enhancements from running specific jobs on VDC resources, as our three policies dictate. The simulator uses the required *.csv* files to make a time range to loop over while checking job times. After initializing variables and running the main simulation loop, statistics are computed and reported in detailed output, and a *.csv* file is generated with the simulation's instantaneous throughput for each runtime second.

3.1.1 Bursting Simulation Loop. The main loop iterates through each second of a DAGMan run analyzing OSG job times to detect completion. While doing so, it implements our policies by checking DAGMan throughput and inspecting submission/execution times to assess OSG's queue and frequency of job submissions. Regarding simulated bursted jobs, for each second in the main loop, we examine each simulated VDC job and increment their tracked runtime by *1* second unless it meets our simulated completion time, which remains constant for both rupture and waveform jobs at *287* and *144* seconds, respectively. In all cases of simulated job completion (VDC and OSG), we have a variable to keep track of overall job completion that is used with the runtime of the loop to calculate instant throughput via (5).

3.1.2 *OSG-Tailored Job Bursting Policies.*

- **Policy 1:** To address low throughput in general, we periodically evaluate if the instant throughput of the DAGMan batch falls below a set threshold. If so, we burst the last unsubmitted OSG job for the phase.
- **Policy 2:** To address congested queues, we regularly analyze submitted OSG jobs and assess their queue duration. If a job has been waiting longer than desired, we remove it from the queue and burst it.
- **Policy 3:** To address gaps in job submissions, we monitor the timestamp of the most recent job added to the OSPool's queue. If there has been more time than desired since then, we periodically burst the last unsubmitted job in the phase.

4 EXPERIMENTAL EVALUATION

For experiments on the FDW, we ran FQs simulations in the Chilean subduction zone, utilizing geometry data from the U.S. Geological Survey project by Hayes et al. [12]. Chile's suitability as a testing ground lies in its record of at least five large earthquake events captured by a dense network of over 120 operating GNSS stations since 2010 [14], which allows for validating and comparing simulated ruptures and waveforms against authentic events. All experiments were designed to recycle the necessary, large matrices (some derived from a required input list of 120+ GNSS stations) in simulations, running with consistent parameters using MudPy's default settings from the GitHub repository. The FDW source code and simulation configurations are openly available at [5] for reproducibility.

4.1 Increasing Earthquake Simulation Quantities

First, we ran simulations in increasing quantities with the FDW, running three DAGMans for each quantity to calculate runtime/total throughput averages and standard deviations (SD(s)). We explored six waveform quantities: 1,024, 2,000, 5,120, 10,000, 24,960, and 50,000, comparable to past work producing 36,800 synthetic FQs waveforms on a single machine [14]. For each quantity, we tested two sizes of an input GNSS station list: one used a full list with 121 stations (*full Chilean input*), and the other used 2 (*small Chilean input*). In this experiment, we aimed to gain insight into the runtime (hours), throughput (jobs/min), and result variability of the FDW to assess OSG's feasibility in accelerating FQs and parallelizing tens of thousands of simulations.

To compute averages for total runtime (α), we summed the three runtimes collected (r_1, r_2, r_3) for each quantity of scenarios and divided the sum by 3 as seen in (1):

$$(r_1 + r_2 + r_3)/3 = \alpha \tag{1}$$

To compute throughput, we divided the number of OSG jobs in each DAGMan (j_n) by its runtime (r_n). This process was done three times for each rupture amount, and then an average was taken to get the average total throughput (β) as follows:

$$((j_1/r_1) + (j_2/r_2) + (j_3/r_3))/3 = \beta \tag{2}$$

4.2 Concurrent HTCondor DAGMans

In optimizing the FDW, we compared the performance of a single DAGMan generating 16,000 waveforms and when two, four, or eight workflows launch simultaneously to create 16,000 waveforms together (using only the full Chilean input). Simulations were configured as described in Section 4. We looked at the same statistics as in Section 4.1: average total runtime, throughput, and the datasets' SDs and maximums/minimums. We also looked at individual jobs' execution and wait times (minutes), instant throughput (jobs/minute), and the number of running jobs during a workflow. This experiment contributes to a deeper understanding of the OSG's behavior, facilitating optimization efforts in accelerating FQs simulations and other tools utilizing HTCondor DAGMan workflows.

The average total runtime (α) was calculated by summing each DAGMan's runtime (d_i) within the parallel batches and dividing that sum by the number of DAGMans (N) it took to create 16,000 rupture scenarios three times:

$$(\sum d_i)/N = \alpha \tag{3}$$

The average total throughputs (β) were calculated by first dividing the number of jobs in each DAGMan (j_i) by its total runtime (r_i). Then, for each different number of DAGMans running in parallel, we summed their total throughputs and divided by the number of batches (N) it took to create 16,000 rupture scenarios three times as shown in (4):

$$(\sum (j_i/r_i))/N = \beta \tag{4}$$

Instant throughput (ω) is the number of complete jobs (j) divided by the current runtime (m) in minutes, as seen in (5):

$$\omega = j/m \tag{5}$$

4.3 Simulated VDC Bursting

This experiment used job times from two actual DAGMan batches to explore parameter variations in two bursting policies. The DAGMan job times used come from Section 4.2 where single DAGMan batches produced 16,000 waveforms. We ran VDC bursting simulations evaluating instant throughput with different probe times (1, 2, 5, 10, 30, 60, 120 seconds) against a 34 jobs/minute threshold (*Policy 1*), preventing job-offloading until the threshold was met, with maximum wait times of 90 and 120 minutes on OSG's queue until bursting (*Policy 2*). To compare the simulated bursting results with actual OSG performance, we used the original time performance from the two DAGMans as a control. The objective was to validate our hypothesis that simulated VDC bursting would reduce total runtime in the FDW and help it to achieve higher, more consistent throughput while maintaining a reasonable percentage of bursted jobs (not more than 30%). This experiment further supported the optimization efforts of the FDW and aimed to demonstrate that supplemental job bursting to federated CI can enhance baseline OSG performance.

We looked at the average instant throughput (jobs/minute) (α), which is the sum of the instant throughputs for every second of the bursting simulation (r_n) divided by the number of simulation seconds (N):

$$(\sum r_n)/N = \alpha \tag{6}$$

We also noted the total runtimes of differing simulations, the maximums/minimums and SDs of the datasets, and the percentage of VDC resources used compared to OSG.

We simulated a bursting cost (USD) (δ) by multiplying the number of simulated VDC minutes used (Cm) by the cost-per-minute (c) for utilized Cloud resources as seen in (7):

$$Cm * c = \delta \tag{7}$$

In this experiment, we used a single cost of 0.0017$ per minute for Cloud computing based on Amazon EC2 on-demand pricing for an a1.xlarge instance with *4 CPUs* and *8GB* of *memory* (which should satisfy our needs) [7].

5 EXPERIMENTAL RESULTS

5.1 Increasing Earthquake Simulation Quantities Results

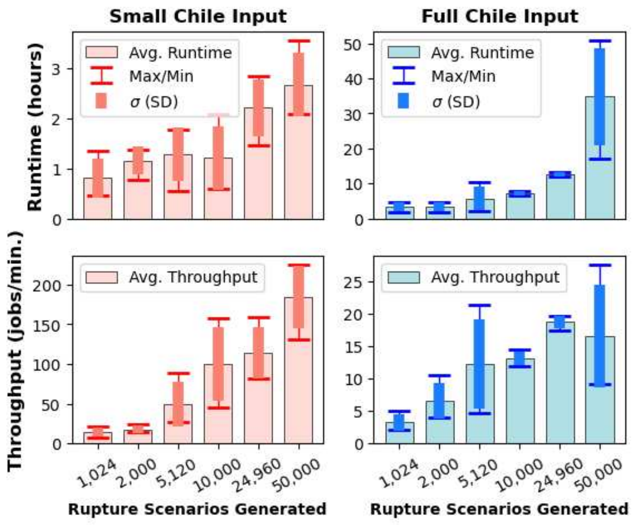

Figure 2: Increasing earthquake simulation quantities experiment statistics. These graphs illustrate the average total runtime and throughput of the FDW simulating varying amounts of earthquake scenarios using two different-sized input lists (2 and 121 stations).

5.1.1 Average Total Runtimes. The average total runtime of workflows using the small Chilean input ranged from 0.8 hours (1,024 waveforms) to 2.7 hours (50,000 waveforms), with an increase of 230.9% in runtime and 4,782.8% in the number of waveforms generated (rounded to 1 decimal place throughout this paper). For workflows using the full Chilean input, the average runtime ranged from 3.3 hours (2,000 waveforms) to 34.8 hours (50,000 waveforms), showing a time increase of 940.5% and a 2,400% increase in simulated scenarios. Although the number of simulations approximately doubled with each increase, the runtime did not follow a directly proportional relationship for both input sizes. The average runtime did not double except at 50,000 waveforms, where it increased by 178% compared to generating 24,960 with the full input (Fig. 2). The dataset with 50,000 ruptures using the full input had the widest range (33.4 hours) and the highest SD (13.9). SDs for all full input scenarios were below 1.2 hours, except for 5,120 and 50,000, and all small input scenarios had average runtime SDs below 1 hour.

5.1.2 Average Total Throughput. With the small Chilean input, the average total throughput ranged from 14.6 jobs per minute (JPM) (1,024 waveforms) to 185 JPM (50,000 waveforms), increasing by 1,165.5%. Using the complete Chilean input with the FDW, throughput ranged from 3.3 JPM (1,024 waveforms) to 18.8 JPM (24,960 waveforms), increasing by 470.2%. When generating 50,000 waveforms, the throughput declined to 16.6 JPM but still exceeded the value when producing 10,000 (13.1 JPM) with the complete input. Notably, the SDs in throughput were significantly lower in the full Chilean input scenarios than in the small ones, as depicted in Fig. 2.

5.2 Concurrent HTCondor DAGMan Results

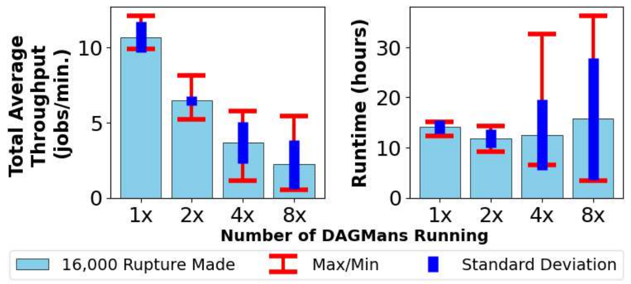

Figure 3: Concurrent HTCondor DAGMan statistics. This figure illustrates the average total throughput and runtime of DAGMan workflows that ran parallel to create 16,000 ruptures using the full Chilean input.

5.2.1 Average Total Throughput. Increasing the number of concurrently running DAGMans resulted in a decrease in throughput for individual DAGMans. When one DAGMan was running, the average total throughput was 10.7 JPM to simulate 16,000 waveforms; for two running, both averaged 6.5 JPM; for four, the average was 3.7; and for eight, it was 2.2 JPM. The average total throughput consistently decreased by at least 39.5% as the DAGMan concurrency level increased (see Fig. 3). When a single DAGMan ran, the throughput was 381.3% better than when eight did.

5.2.2 Average Total Runtime. Although the number of waveforms generated by individual DAGMans decreased as more ran concurrently, the average total runtime for individual DAGMans did not decrease proportionately. For instance, when eight DAGMans were running to create 16,000 waveforms (each making 2,000), their average total runtime was slower than when a single DAGMan produced 16,000. The average total runtime for a single workflow was 14.1 hours (SD 1.3); when two, four, and eight DAGMans ran simultaneously, they were 11.9 (SD 1.8), 12.5 (SD 7), and 15.7 (SD 12) hours, respectively.

5.2.3 Job Execution and Wait Times. The execution times of FDW jobs remained consistent across all experiments (4.1 and 4.2). Jobs simulating waveforms with the full 121-station list typically took 15 to 20 minutes, whereas those using two stations often completed in

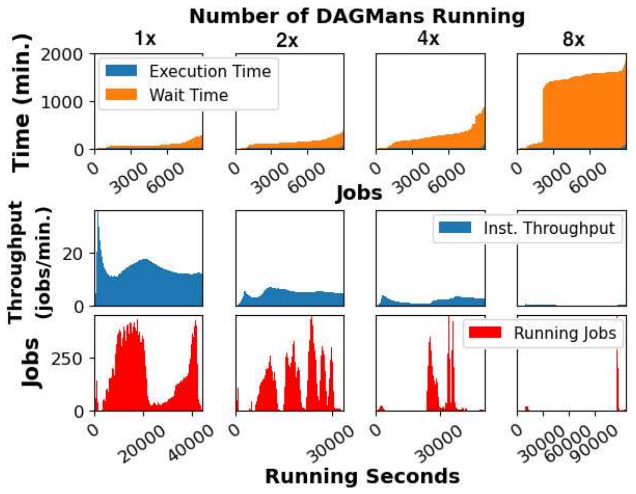

Figure 4: Concurrent DAGMan statistics continued. These graphs showcase examples of individual jobs' execution and wait times (sorted by duration) and instant throughput and running job count (for every running second of individual workflows) in various numbers of concurrently running DAGMans.

under 1 minute. Jobs simulating ruptures consistently executed in around 2.5 minutes across all experiments. Overall, job wait times varied, ranging from multiple hours to seconds. When multiple DAGMans ran concurrently, there was a higher frequency of jobs with long wait times (refer to Fig. 4). For example, with four DAGMans, the average wait time for waveform jobs was 189.2 minutes, whereas, with one, it was 70.1 minutes.

5.2.4 Instant Throughput and Running Jobs. The instant throughput of running DAGMans exhibited unpredictability, except for a significant initial spike caused by the A Phase, comprising fewer and shorter jobs than C. These peaks were notably lower when more DAGMans ran simultaneously. For instance, when four ran in parallel, their peaks rarely exceeded 6 JPM, whereas lone batches reached heights of over 35, as seen in Fig. 4. Notably, more concurrent DAGMans led to lower instant throughput on average.

In Fig. 4, the running job footprints exhibited erratic behavior across all batches. Increasing the number of concurrent DAGMans resulted in more gaps and running job peaks, possibly due to a decrease in the number of jobs per DAGMan. However, similarities were observed in running job peaks between single and multiple batch executions. For example, all sizes of DAGMans running (1, 2, 4, and 8) had instances with over 400 running jobs (see Fig. 4). Although running job footprints in DAGMan runs differed significantly from throughput, there was a correlation between their peaks/dips when the OSG launched job groups.

5.3 Simulated VDC Bursting Results

5.3.1 Average Instant Throughput. In our VDC bursting simulation, the probing of *Policy 1* influenced the average instant throughput (AIT), as seen in Fig. 5 - faster probe times for bursting increased instant throughput and throughput variability. The control (representing OSG performance) had the lowest AIT: 14.1 JPM (*Batch 1*) and 8.6 JPM (*Batch 2*); the maximums were 31.7 and 32.4 JPM for batches *1* and *2*, respectively, with a probe time of 1 second and an allowed queue time of 90 minutes Within each batch, the maximum and minimum average throughputs remained consistent. The minimum was always 0 JPM before job completion, whereas short jobs in the A Phase influenced the maximum. The throughput remained similar for differing queue times before bursting OSG jobs in *Policy 2*. A 30-minute shorter maximum queue time resulted in more bursted jobs and a slight increase in instant throughput for both batches. However, the difference never increased the AIT by more than 1 JPM.

5.3.2 Cloud/VDC Usage Compared to OSG. Similarly, the percentage of Cloud/VDC usage was dependent on *Policy 1* and not on the other two policies; when the probe time shortens in seconds, it leads to higher VDC utilization. For *Batch 1* and *Batch 2*, the maximum simulated Cloud usages were 52.8% and 85.6%, respectively, whereas the minimums were 19.1% and 22.9%, respectively. *Batch 2* had more Cloud usage than *Batch 1* because its execution was longer than that of 1, allowing more time for probing and bursting. The increased use is more apparent when the probe time is less than 10 seconds, as seen in Fig. 5.

5.3.3 Runtime. The results of the total runtimes of the VDC bursting simulations were similar to that of the AIT; they were much more influenced by *Policy 1* than *2* and *3*. In some cases, although the batch's AIT increased in the simulator over the original OSG performance, the runtime stayed similar. For example, in *Batch 2*, with a 90-minute allowed queue time and a 1-minute probe time, the AIT improved by approximately 22%, but the runtime decreased by less than one minute (see Fig. 6). Simulated VDC bursting led to reduced runtimes in *Batch 1*, with some cases experiencing multiple hours of improvement, for instance, when using a 10-second probe and 120-minute queue time.

5.3.4 Cost. In the simulations, we spent up to $11 for *Batch 1* and $13.9 for *Batch 2* to help generate 16,000 waveforms, ensuring that no more than 30% of jobs were bursted. The probe time in *Policy 1* and the runtime of the batch had the most significant influence on cost.

6 DISCUSSION

Using the FDW, we generated 1,024 waveforms with the full Chilean input in less than half the time compared to running an automated version of MudPy's FakeQuakes on a single host (the Amazon AWS instance described in 3.1) with the same simulation parameters. Parallel computation exhibited a powerful effect, significantly reducing execution time and increasing throughput as the number of simulations increased. The FDW excelled in doing tens of thousands of simulations, a challenge for Lin et al. [14]. In contrast to their over-20-day generation of 36,800 waveforms, we produced, on average, 24,960 in 12.5 hours and 50,000 in under 35 hours. Although the FDW generally exhibited low variability, with 50,000 waveforms using the full input, significant volatility was observed, likely due to OSG's variable resources and many simulations. The SDs of the throughput datasets were much greater when using the

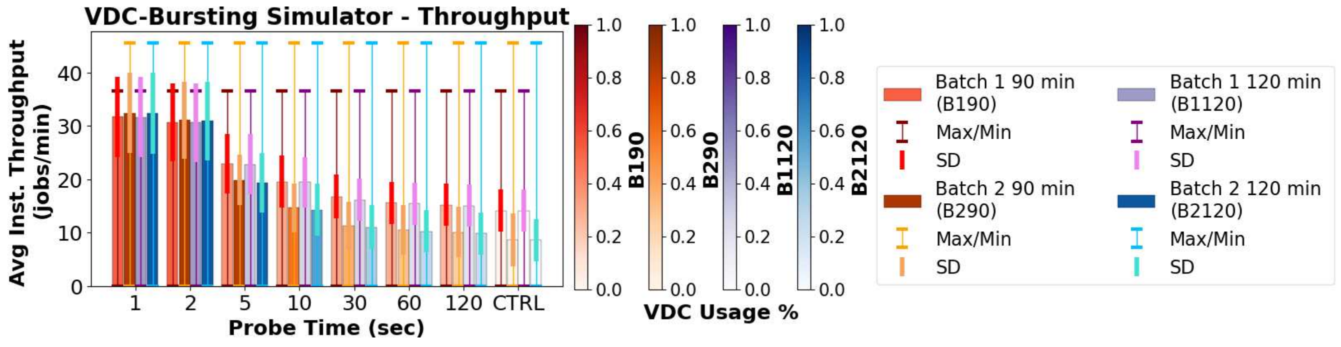

Figure 5: VDC bursting experiment statistics. This graph illustrates the average instant throughputs and VDC utilization percentages while simulating supplemental job bursting for OSG jobs. We investigated two policies: the first evaluated against a throughput threshold with varying probe times for bursting, and the second examined different maximum allowed queue times until VDC bursting jobs.

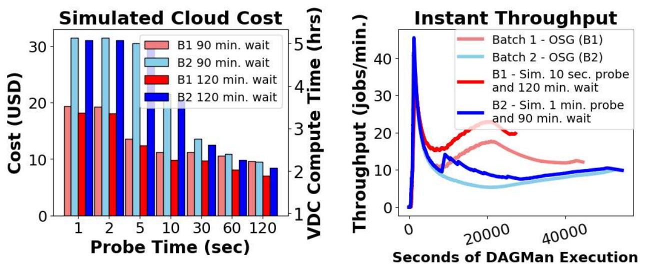

Figure 6: VDC bursting results continued. These graphs showcase the simulated cost (left) for supplemental VDC bursting on two real OSG DAGMans examples and their instant throughput over time (right).

full input compared to the small. Thus, the FDW's performance is less variable for more complex simulations, which is desirable.

Our experiment investigating the impact of splitting the creation of 16,000 waveforms into multiple DAGMans compared to a single one revealed that OSG performs best with a single running DAGMan. Increasing the number of simultaneous running batches leads to decreased throughput, and reducing the number of jobs in each batch does not result in the expected decrease in runtime, as seen in Fig. 3. Also, as the number of concurrently running DAGMans increases, the SDs grow, indicating increasing result instability. Although varying resource availability might influence some outcomes, the overall trend is evident: partitioning workloads into multiple simultaneously running DAGMans is not advantageous on the OSG.

Our VDC bursting simulator improved FDW runtime and throughput in all experiments, with varying performance across different batches. Notably, *Batch 2* showed negligible runtime reduction when the VDC bursted job limit was reached. However, *Batch 1* demonstrated promising results with a substantial 38.7% decrease in simulated workflow runtime while staying within our desired bursted job limit, as seen in Fig. 6. Additionally, simulated throughput SDs deteriorated compared to OSG. Hence, there is room for improvement to achieve consistent throughput in real-world implementations. Based on the simulation, the experimental evaluation is an initial stage in creating a comprehensive, elastic algorithm for bursting OSG jobs to VDC resources. By enhancing policy dynamics, we aim to achieve similar execution times for FDW workloads and improve OSG workflow behaviors by scaling utilized VDC resources based on OSG's common resources. Although our bursting simulator showed promising results in one of the two tested batches, our policies (especially *Policy 1*) demonstrate an ability to enhance the OSG with federated CI resources.

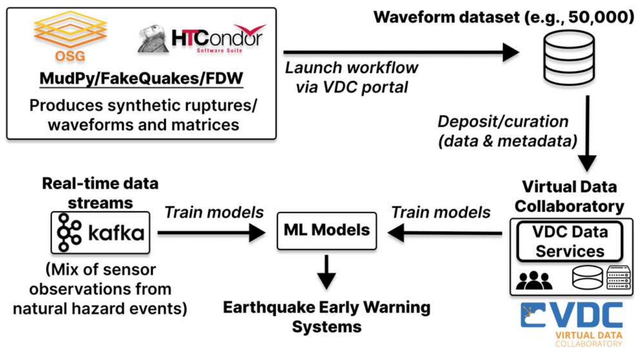
OSG
HTCondor
MudPy/FakeQuakes/FDW
Produces synthetic ruptures/ waveforms and matrices
Launch workflow via VDC portal
Waveform dataset (e.g., 50,000)
Deposit/curation (data & metadata)
Virtual Data Collaboratory
VDC Data Services
Train models
ML Models
Real-time data streams
kafka
(Mix of sensor observations from natural hazard events)
Earthquake Early Warning Systems
VDC
VIRTUAL DATA COLLABORATORY

Figure 7: Workflow for accelerating seismic research. This figure depicts the flow of simulated data from the FDW to the VDC and beyond. VDC provides means for curating FDW data, which can be retrieved and used by earthquake early warning models and more.

In this paper, we lay the groundwork for integrating the MudPy synthetic data toolkit into the VDC and implementing an automated system for bursting OSG jobs to VDC Cloud resources based on user-defined policies. We aim to optimize simulations further, promote tool democratization and data sharing, and expand VDC data services. The VDC serves to enhance MudPy by providing a GUI-based platform for executing accelerated simulations and monitoring their progress, providing equitable access to MudPy for researchers of all backgrounds. Furthermore, VDC data services enable data deposition, curation, and tagging with metadata, allowing synthetic data products to be accessed more easily and timely for training EEW models, as illustrated in Fig. 7. Large datasets will be able to be efficiently distributed via optimized caching systems and even prefetched for users via AI-based "intelligent data delivery services" [25] that utilize user query traces and institutional data, thus providing opportunities for accelerated research.

7 CONCLUSION AND FUTURE WORK

This paper leverages the NSF-funded OSG to parallelize MudPy's earthquake simulations and accelerate their execution; using our workflow, we observed a significant reduction in execution time and increased throughput for tens of thousands of simulations. Optimization insights into HTCondor DAGMans using our workflow demonstrate higher throughput when running a single DAGMan versus running multiple concurrently (e.g., eight) to execute a workload. Additionally, we address OSG's shared resource issues through our VDC bursting simulation framework and three policies that respond to long waits, gaps in job submissions, and low throughput. Further elaboration on our VDC bursting model will be required. This work is the first step in integrating MudPy and OSG into the VDC ecosystem. Our overarching goal is democratizing and facilitating efficient data access and integrating scientific data sources with diverse research communities. Experience gained from developing the workflow and Cloud/VDC bursting simulator described in this paper can apply to other data services. Efficiently made AI-ready products from the FDW, curated by the VDC, can accelerate interdisciplinary scientific research to fully unleash the potential of data and AI in science workflows for significant societal impacts (e.g., earthquake early warning). Future work includes experimenting with regions beyond Chile, helping researchers leverage and modify existing datasets and create new meaningful products, and exploring HTC workflows and VDC bursting policies for other use cases and CI configurations.

ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under grants OAC 2219975 and OAC 2220826 and was done using services provided by the OSG Consortium [1, 24, 27], which is supported by the National Science Foundation awards OAC 2030508 and OAC 1836650.

REFERENCES

[1] 2006. Open science pool. https://doi.org/10.21231/906P-4D78
[2] 2023. *Apptainer.* Retrieved May 23, 2023 from https://apptainer.org/
[3] 2023. *Conda.* Retrieved May 23, 2023 from https://docs.conda.io/en/latest/
[4] 2023. *HTCondor.* Retrieved May 23, 2023 from https://htcondor.org/
[5] Marcus Adair. 2023. *FDW Source Code.* https://github.com/Marcus-Adair/Accelerating-Data-Intensive-Seismic-Research-Through-Parallel-Workflow-Optimization-and-Federated-CI
[6] Richard M. Allen and Diego Melgar. 2019. Earthquake Early Warning: Advances, Scientific Challenges, and Societal Needs. *Annual Review of Earth and Planetary Sciences* 47, 1 (May 2019), 361–388. https://doi.org/10.1146/annurev-earth-053018-060457
[7] Amazon. 2023. *Amazon EC2 On-Demand Pricing.* Retrieved May 23, 2023 from https://aws.amazon.com/ec2/pricing/on-demand/
[8] Marc C. Carrillo. 2023. *Paral·lelització del codi del simulador de terratrèmols MudPy amb CUDA.* Master's thesis. Dept. Computer Science, Open Univ. of Catalonia, Barcelona, Spain, June 2021.
[9] Ewa Deelman, Karan Vahi, Mats Rynge, Rajiv Mayani, Rafael Ferreira da Silva, George Papadimitriou, and Miron Livny. 2019. The Evolution of the Pegasus Workflow Management Software. *Computing in Science & Engineering* 21, 4 (2019), 22–36. https://doi.org/10.1109/MCSE.2019.2919690
[10] Kevin Fauvel, Daniel Balouek-Thomert, Diego Melgar, Pedro Silva, Anthony Simonet, Gabriel Antoniu, Alexandru Costan, Véronique Masson, Manish Parashar, Ivan Rodero, et al. 2020. A distributed multi-sensor machine learning approach to earthquake early warning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 403–411. https://doi.org/10.1609/aaai.v34i01.5376
[11] Dara E. Goldberg and Diego Melgar. 2020. Generation and Validation of Broadband Synthetic P Waves in Semistochastic Models of Large Earthquakes. *Bulletin of the Seismological Society of America* 110, 4 (07 2020), 1982–1995. https://doi.org/10.1785/0120200049 arXiv:https://pubs.geoscienceworld.org/ssa/bssa/article-pdf/110/4/1982/5114114/bssa-2020049.1.pdf
[12] Gavin P. Hayes, Ginevra L. Moore, Daniel E. Portner, Mike Hearne, et al. 2018. Slab2, a comprehensive subduction zone geometry model. *Science* 362, 6410 (2018), 58–61. https://doi.org/10.1126/science.aat4723
[13] HTCondor. 2023. *Dagman workflows.* Retrieved August 1, 2023 from https://htcondor.readthedocs.io/en/latest/automated-workflows/index.html?highlight=DAGMan
[14] Jiun-Ting Lin, Diego Melgar, Amanda M. Thomas, and Jacob Searcy. 2021. Early Warning for Great Earthquakes From Characterization of Crustal Deformation Patterns With Deep Learning. *Journal of Geophysical Research: Solid Earth* 126, 10 (September 2021). https://doi.org/10.1029/2021JB022703
[15] Michael Mattess, Christian Vecchiola, Saurabh Garg, and Rajkumar Buyya. 2017. Cloud Bursting: Managing Peak Loads by Leasing Public Cloud Services. *Cloud Computing: Methodology, Systems, and Applications* (January 2017), 343–367.
[16] Diego Melgar. 2019. *Stochastic slip (fakequakes).* Retrieved May 23, 2023 from https://github.com/dmelgarm/MudPy/wiki/Stochastic-slip-(fakequakes)
[17] Diego Melgar. 2022. *MudPy.* Retrieved May 19, 2022 from https://github.com/dmelgarm/MudPy
[18] Diego Melgar, Brendan W. Crowell, Timothy I. Melbourne, Walter Szeliga, Marcelo Santillan, and Craig Scrivner. 2020. Noise Characteristics of Operational Real-Time High-Rate GNSS Positions in a Large Aperture Network. *Journal of Geophysical Research: Solid Earth* 125, 7 (June 2020), e2019JB019197. https://doi.org/10.1029/2019JB019197
[19] Diego Melgar and Gavin P. Hayes. 2019. Characterizing large earthquakes before rupture is complete. *Science Advances* 5, 5 (May 2019). https://doi.org/10.1126/sciadv.aav2032
[20] Diego Melgar, Randall J. LeVeque, Douglas S. Dreger, and Richard M. Allen. 2016. Kinematic rupture scenarios and synthetic displacement data: An example application to the Cascadia subduction zone. *Journal of Geophysical Research: Solid Earth* 121, 9 (August 2016), 6658–6674. https://doi.org/10.1002/2016JB013314
[21] OSG. 2023. *Computation on the open science pool.* Retrieved May 23, 2023 from https://portal.osg-htc.org/documentation/overview/account_setup/is-it-for-you/
[22] OSG. 2023. *Running OSDF cache in a container.* Retrieved May 23, 2023 from https://osg-htc.org/docs/data/stashcache/run-stashcache-container/
[23] Manish Parashar, Anthony Simonet, Ivan Rodero, Forough Ghahramani, Grace Agnew, Ron Jantz, and Vasant Honavar. 2020. The Virtual Data Collaboratory: A Regional Cyberinfrastructure for Collaborative Data-Driven Research. *Computing in Science & Engineering* 22, 3 (June 2020), 79–92. https://doi.org/10.1109/MCSE.2019.2908850
[24] Ruth Pordes, Don Petravick, Bill Kramer, Doug Olson, et al. 2007. The open science grid. In *J. Phys. Conf. Ser. (78, Vol. 78).* 012057. https://doi.org/10.1088/1742-6596/78/1/012057
[25] Yubo Qin, Ivan Rodero, and Manish Parashar. 2022. Toward Democratizing Access to Facilities Data: A Framework for Intelligent Data Discovery and Delivery. *Computing in Science & Engineering* 24, 3 (2022), 52–60. https://doi.org/10.1109/MCSE.2022.3179408
[26] Christine J. Ruhl, Diego Melgar, Ronni Grapenthin, and Richard M. Allen. 2017. The value of real-time GNSS to earthquake early warning. *Geophysical Research Letters* 44, 16 (August 2017), 8311–8319. https://doi.org/10.1002/2017GL074502
[27] Igor Sfiligoi, Daniel C Bradley, Burt Holzman, Parag Mhashilkar, Sanjay Padhi, and Frank Wurthwein. 2009. The pilot way to grid resources using glideinWMS. In *2009 WRI World Congress on Computer Science and Information Engineering (2, Vol. 2).* 428–432. https://doi.org/10.1109/CSIE.2009.950
[28] Igor Sfiligoi, Michael Hare, David Schultz, Frank Würthwein, Benedikt Riedel, Tom Hutton, Steve Barnet, and Vladimir Brik. 2021. Managing Cloud Networking Costs for Data-Intensive Applications by Provisioning Dedicated Network Links. In *Practice and Experience in Advanced Research Computing* (Boston, MA, USA) *(PEARC '21).* Association for Computing Machinery, New York, NY, USA, Article 18, 8 pages. https://doi.org/10.1145/3437359.3465563
[29] Igor Sfiligoi, David Schultz, Frank Würthwein, and Benedikt Riedel. 2021. Pushing the Cloud Limits in Support of IceCube Science. *IEEE Internet Computing* 25, 1 (2021), 71–75. https://doi.org/10.1109/MIC.2020.3045209
[30] U.S. Geological Survey. 2023. *Search earthquake catalog.* Retrieved June 19, 2023 from https://earthquake.usgs.gov/earthquakes/search/