Job and Data Clustering for Aggregate Use of Multiple Production Cyberinfrastructures

Ketan Maheshwari
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL USA
ketan@mcs.anl.gov

Allan Espinosa
Department of Computer Science
University of Chicago
Chicago, IL USA
aespinosa@cs.uchicago.edu

Daniel S. Katz
Computation Institute
University of Chicago & Argonne National Laboratory
Chicago, IL USA
d.katz@ieee.org

Michael Wilde
Computation Institute
University of Chicago & Argonne National Laboratory
Chicago, IL USA
wilde@mcs.anl.gov

Zhao Zhang
Computation Institute
University of Chicago
Chicago, IL USA
zhaozhang@uchicago.edu

Ian Foster
Mathematics and Computer Science Division
Argonne National Laboratory
Argonne, IL USA
foster@mcs.anl.gov

Scott Callaghan
Southern California Earthquake Center
University of Southern California
Los Angeles, CA USA
scottcal@usc.edu

Phillip Maechling
Southern California Earthquake Center
University of Southern California
Los Angeles, CA USA
maechlin@usc.edu

ABSTRACT

In this paper, we address the challenges of reducing the time-to-solution of the data intensive earthquake simulation workflow "CyberShake" by supplementing the high-performance parallel computing (HPC) resources on which it typically runs with distributed, heterogeneous resources that can be obtained opportunistically from grids and clouds. We seek to minimize time to solution by maximizing the amount of work that can be efficiently done on the distributed resources. We identify data movement as the main bottleneck in effectively utilizing the combined local and distributed resources. We address this by analyzing the I/O characteristics of the application, processor acquisition rate (from a pilot-job service), and the data movement throughput of the infrastructure. With these factors in mind, we explore a combination of strategies including partitioning of computation (over HPC and distributed resources) and job clustering.

We validate our approach with a theoretical study and with preliminary measurements on the Ranger HPC system and distributed Open Science Grid resources. More complete performance results will be presented in the final submission of this paper.

Categories and Subject Descriptors

J.2 [**Computer Applications**]: Earth and atmospheric sciences; Engineering—*SCR*

General Terms

Theory and Implementation

Keywords

swift, parallel, hpc, scec

1. INTRODUCTION

Our work aggregates XSEDE[1] and Open Science Grid (OSG) to run the CyberShake application [4,12] faster than on XSEDE alone. XSEDE is most often used as a distributed collection of high-performance systems, where small numbers of large parallel jobs are run, usually each on a single system. OSG is a distributed collection of resources that is most often used for large number of small, high-throughput, location-independent jobs. While these two infrastructures can both be accessed through common grid interfaces (e.g., GRAM to submit jobs, GridFTP to move data), they are disjoint in terms of access procedures, operations, support, and the types of jobs (e.g., parallel vs. serial) for which their resources are optimized. This results in lost

[1]We use XSEDE here to refer to the TACC Ranger HPC system, wide area network, etc., which were called TeraGrid when this work started and since have transitioned into XSEDE, the "Extreme Science and Engineering Discovery Environment".

opportunities for resource utilization, as most scientists only use one grid or the other. Using the Swift parallel scripting framework [24] to express computations independently of the resources used, we show how CyberShake can exploit an expanded set of resources across both XSEDE and OSG.

As we will further discuss, using Swift for CyberShake provides several advantages, in particular:

1. A seamless and transparent interface to schedulers on two different execution sites, SGE on XSEDE and GWMS on OSG;
2. Transparent linking and execution of serial and parallel stages of the workflow;
3. Sophisticated data management, including through Coasters;
4. Exploitation of the enormous task level parallelism of post-processing stages, *get_variations*, *extract*, and *seispeak*; and
5. Fault tolerant execution via retry and resume features,

Items 2, 4, and 5 were also provided by the original Pegasus implementation of the application as a pair of workflows. Item 1 was also provided by Pegasus, but only for a single site on OSG. We chose to use Swift for this work primarily because of the potential gains in item 3 and the fact that we believe the script version of the application is both easier to understand and to modify than the DAG version of the application.

Our initial implementation of the computation showed data-intensive properties that causes inefficiencies when using both cyberinfrastructures [9]. We describe here the process by which we approached the problem of scaling CyberShake [10]. We profiled the application on XSEDE and OSG and identified several approaches to exploit the combination of the two different cyberinfrastructures.

2. CYBERSHAKE

The Southern California Earthquake Center (SCEC) uses a software platform called CyberShake to calculate probabilistic ground motion. This method characterizes earthquakes by quantifying peak ground motions from all possible earthquakes that might affect a particular site. There are two primary computation elements in generating hazard risk curves [15]. First, a list of earthquake (rupture) locations that may occur in a region are produced. Second, at a particular geographic site of interest, the ground level motion produced by these earthquakes is calculated. For each rupture, there are a number of variations of what might occur. We refer to the data describing the variations as *rupture variations.* This dataset is currently updated infrequently, once a year or so.

The core computation component of CyberShake generating all geographic sites' hazard curve. This computation is made in four steps. First, anelastic wave propagation simulation is performed to calculate strain Green tensors (SGTs) around a site of interest[2]. As shown in Figure 1, the SGT generation stage is a linear pipeline of processes consisting of a single critical path. The tasks in this part of the application are dominated by MPI-based parallel tasks. We use a XSEDE HPC system for this stage. The dominant compute-intensive process in this stage is a tightly-coupled parallel task used to generate the SGT dataset ($simSGT$). The task uses 800 cores for approximately 20 hours to create the fragments for the x and y components of the SGT mesh. These fragments are merged ($mergeSGT$) into the corresponding components and are saved in two individual files, each of size 19.8 GB.

The last three stages (*get_variations, extract, and seispeak*) of the hazard curve generation process are loosely-coupled in nature, and in combination with the two preceding database query stages: *get_site, get_ruptures*, we refer to them as "post-processing". Figure 2 shows an overview of the post-processing, including the data consumed and produced by each stage. This portion of the computation is pleasingly parallel, including both parallel processing across ruptures and parallel processing across variations within a rupture.

Subcomponents of the SGTs relevant to each rupture (called "subSGT") are first obtained by *extract* [15]. The LGU site can be affected by 5,939 earthquakes. This corresponds to 5,939 *extract* jobs in the post-processing workflow that generates subSGT components. These jobs require the x and y components of the SGT mesh (39.6 GB) and earthquake rupture information from the ERF (864.14 kB) as inputs, and produce the subcomponents (163.43 MB) of the SGT mesh corresponding to a rupture as the output.

The next step is seismogram synthesis using seismic reciprocity [27]. This task is performed for each rupture variation. Run times of each *seismogram* task are 1-30 min. For the 5,939 earthquakes, there are a total of 403,852 rupture variations. This makes the post-processing component of the hazard curve generation data intensive. Each *seismogram* needs the x and y subcomponents of the SGT for the rupture (163.43 MB) and one of the variation of the associated earthquake (4.09 MB), and each produces 24 kB of output, totaling 9.72 GB of output. Coupled with generating each seismogram is calculating peak spectral acceleration (*peak_calc*) of the seismogram. Each *peak_calc* job outputs 214 B, and the total of all their outputs is 86.72 MB. We bundle the *seimogram* and *peak_calc* jobs together into a job called *seispeak*. The nature of *seispeak* jobs is such that a group of them may require a common SGT for the rupture. As a result a large number of them could be grouped into a single job and sent to the target execution site with the common SGT data, thus saving redundant data movement.

Overall, the characteristics of CyberShake are shown in Table 1, including:

- diverse computational requirements of time and CPUs;
- the *preSGT* stage requires 16,000 core hours while *postproc* requires 48,000 core hours;
- diverse data management requirements, ranging from a few kB to many GB; and
- the number of parallel instances for each stage vary from one to hundreds of thousands.

3. CYBERINFRASTRUCTURES

In this work, we leverage both Open Science Grid (OSG) and XSEDE by distributing tasks suited for each of these infrastructures. We use XSEDE for the SGT generation,

[2]In this paper, we use Laguna Peak (LGU) unless otherwise specified.

	Task	Serial/Parallel	# Instances	Avg. runtime per instance	Input Data per instance	Output Data per instance
SGT-Generation	PreCVM	serial	1	30 min	0.1 MB	42 MB
	PreSGT	parallel	1	30 min, 16 cores	42 MB	30 MB
	VMeshGen	parallel	1	120 min, 160 cores	42 MB	3 GB
	VMeshMerge	parallel	1	60 min, 8 cores	4 GB	12 GB
	SimSGT	parallel	2	20 h, 400 cores	12 GB	20 GB
	MergeSGT	serial	2	120 min	20 GB	19.8 GB
post-processing	getSite	serial	1	1 min	0.5 kB	1.5 kB
	getRuptures	serial	1	4 min	1.5 kB	4 kB
	getVariations	serial	5939	5 min	4 kB	5 MB
	extract	serial	5939	26 s	40 GB	164 MB
	seispeak	serial	403852	3 min	164 MB	25 kB

Table 1: Profile of CyberShake workflow tasks and stages. Figures 1 and 2 show total data sizes.

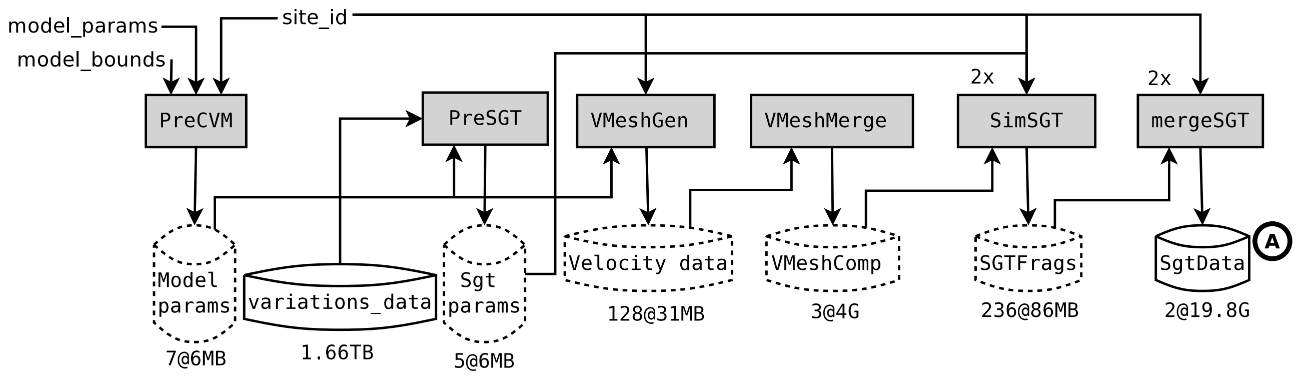

Figure 1: Overview of the SGT-generation workflow for CyberShake. Light gray colored stages are run on XSEDE while the dark gray colored are run partially on OSG and XSEDE. Datasets with dotted outlines represent intermediate data. Note that SgtData, marked with Ⓐ, is the link between the two computation phases.

since there are a relatively small number of jobs in this stage and the compute-intensive jobs are medium-scale (800-core) parallel jobs, and combine use of XSEDE and OSG for the post-processing stage, since it involves a large number of single-core tasks. Here, we briefly describe the two infrastructures, focusing on the characteristics that are crucial for our application and our analysis, and how we have used them.

OSG [17] was established to support the analysis of data from large physics experiments such as the LHC, LIGO, and STAR. In mid-2011, OSG consisted of approximately 131 resource providers, with a total of approximately 70,000 cores. OSG jobs are scheduled on a per-core basis. OSG is based on resource sharing; most resource providers support opportunistic usage of their resources, meaning that they may be used by others when idle.

On average, 20-30% of total OSG compute cycles are available on an opportunistic basis. That does not translate to specific availability at any point in time but serves as an estimate for a sufficiently long period of time. Within this availability, all virtual organizations (VOs) can compete for these cycles based on policies implemented at each site; sites have autonomy on how they setup policy governing which VOs are given access to opportunistic cycles and the relative priority of the VOs for these cycles. One of these VOs is called Engage, and is intended for groups to explore use of OSG. The OSG Engagement group runs this VO and handles support and access for opportunistic accessibility by these groups. Over a 30-day period in late 2011, an average of 2,300 cores have been in use by Engage users, matching the demand of Engage users with no wait time. There were peaks of at least 6,500 cores at particular points in time, providing end users with the desired throughput. Between December 2010 and November 2011, the Engage VO received over 13M hours across OSG [16].

To determine what level of resources OSG can provide to the CyberShake application, we generated a similar workload with the data transfer components removed. The synthesized computation is composed of thousands of 5 min jobs. We ran 6,000 instances of these jobs. Within a 2,000 s period, 1,500 jobs completed, consuming 125 CPU-hours . The acceleration factor of approximately 225 is not enough to finish the CyberShake application within our target time-to-solution of 1–2 days. OSG recommends that jobs have a duration of 4–12 hours for best throughput. We can use multi-level scheduling techniques to run jobs of these lengths as containers that will run the short jobs in the CyberShake application. To test this, we sent 4-hour jobs to 18 OSG resources. Over a period of 24 hours, we obtained a total number of 33,036 CPU-hours, which is well within the CyberShake time-to-solution target of 1–2 days.

The TeraGrid [14, 20] environment, which has now been

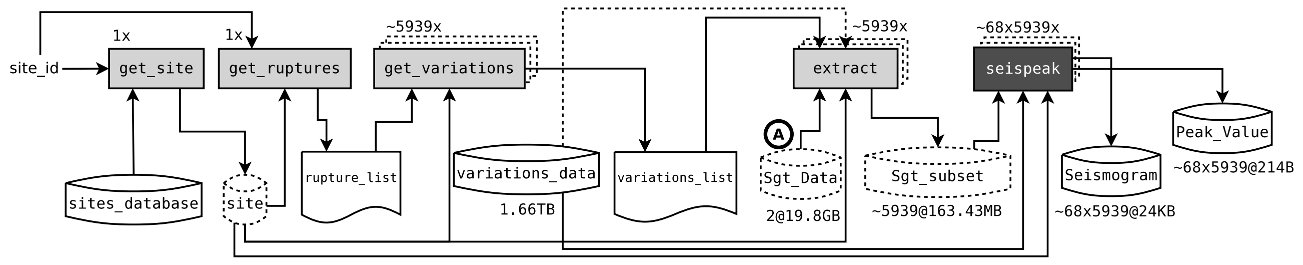

Figure 2: Overview of the post-processing workflow for CyberShake. Light gray colored stages are run on XSEDE while the dark gray colored stage (seispeak) is run partially on OSG and XSEDE. Dashed lines represent partial use of a dataset. Note that SgtData, marked with Ⓐ, is the link between the two computation phases.

superseded by XSEDE [26], was a research computing infrastructure that combined large computing and data management facilities. From its initial capability of 24 Tflops of computing power and 800 TB of fast disk storage, TeraGrid grew to 2 Pflops and 50 PB worth of storage capacity. The facilities in TeraGrid were linked through a 10 Gbit/s optical network. Many TeraGrid systems ran Linux as their operating system. The software stack typically included a wide range of compilers. Various implementations of the Message Passing Interface were also included on top of these compilers. Cluster management tools for submitting and managing jobs were varied. For the sites to communicate, all had an installation of the Globus Toolkit [11], which provided software for inter-site communications such as remote resource execution, data transfer and secure authentication. Our use of XSEDE in this work involves the Ranger HPC system at TACC and the network connecting it to other systems, including OSG.

When this project was started, TeraGrid and OSG were independent infrastructures. Each was separately funded, and each had its own goals. This project is part of a larger project called ExTENCI [3], which aims to help the infrastructures work together to deliver science capabilities to users. More recently, since TeraGrid was superseded by XSEDE, discussions have taken place to tighten the connections between the two infrastructures.

Production CyberShake currently involves running two coupled Pegasus [8] workflows, one for SGT generation and the other for post-processing, for each geographic site to be studied. As SCEC runs CyberShake for a number of sites, it distributes these pairs of workflows across TeraGrid computing resources such as NCSA Abe and TACC Ranger, and computing cluster resources at SCEC/USC, such that each resource runs a number of complete site analyses [4]. For the work described in this paper, we 1) continued to run the SGT generation on XSEDE resources (Ranger) and examined how best to run the post-processing workflow on a combination of XSEDE and OSG, and 2) examined how to automate such decisions for additional applications.

Of particular importance to this project is the network connections between XSEDE and OSG. CyberShake requires hundreds of GB of data movement. Thus, fast data staging rates are needed in order to keep the OSG resources filled with CyberShake jobs.

4. RELATED WORK

As previously stated, the current CyberShake production system runs on XSEDE systems and USC clusters using Pegasus to manage its workflows [4]. While the production runs compute multiple hazard curves, each one is computed on a single large resource. Some experiments have also been done with running the post-processing workflow on OSG [21], but this work only used a single OSG site, so in a sense, it wasn't different than using a single TeraGrid cluster, other than interface issues. In addition, a different application was run using Pegasus on both TeraGrid and multiple OSG systems [21]. In both applications, issues related to data movement and deciding how best to use multiple systems for the different tasks within the processing were not investigated. In contrast, here we investigated how to map and optimize the workflow for generating a single hazard curve performs when executed over multiple compute resources that reside on different cyberinfrastructures that include many physically distributed resources.

Pegasus traditionally maps the entire abstract workflow at once. Resource allocation decisions can be made prematurely because of the dynamic nature of computing resources. Changes in cyberinfrastructure occur often because of resource failure or policy changes. Pegasus tried to address this issue through *just-in-time planning* [7]. The scheme partitions the abstract workflow into series of partial ones. Planning is performed for the next partial workflow just-in-time when the current partial workflow finishes. This is implemented as the last job of a concrete workflow in DAGman.

The SCEC team reported scaling issues in the mapping process of the abstract workflow due to the volume of jobs being produced. Recursive DAGs [6] were used to reduce the number of jobs being submitted to the Condor scheduler. Because of the data-intensive nature of tasks in post-processing, pilot jobs were employed through Condor Glidein-WMS (GWMS) [4, 22].

Job clustering, building a new job that has a number of smaller jobs in it, is used in Pegasus primarily to reduce the number of jobs being sent to the Condor pool handling the GWMS [5]. In this work, we use job clustering for this reason, but also to reduce the redundant transfer of data, as discussed in §7.

Raicu's work on "data diffusion" [18] uses dynamic resource allocation as the demand for data increases. Each allocated computing resource then does data caching to improve locality characteristics of succeeding jobs. Ranganathan proposed active data replication to acquired computing resources to improve performance based on his simulation results [19]. But this technique can only benefit input data that is shared between input tasks. The bulk of the jobs in the CyberShake post-processing computation involves a large number of individual small files.

5. SWIFT

Swift is a parallel scripting language for scientific computing. We use Swift [24] to express both the SGT generation and post-processing parts of the workflow in a single integrated script. The Swift language is typed and concurrent, and it provides multiple features to support distributed scientific batch computing, include data structures (arrays, structs), string processing, use of external programs, and external data access to filesystem structures. The current Swift implementation compiles programs into the Karajan workflow language, which is interpreted by a runtime system based on the Java CoG Kit [23]. Swift is capable of generating and scheduling thousands of tasks, and managing their execution on a wide range of distributed resources. Swift has a pilot job mechanism called Coasters, and there is ongoing research and development in Collective Data Management to optimize I/O.

The Coasters [13] framework is a mechanism that is integrated in the Swift framework. Coasters are based on a service-worker paradigm of distributed computing. A Coaster 'service' manages 'workers' on distributed resources. Coasters run these workers as task-placeholders on target resources, resulting in an opportunistic reservation of resources similar to that of pilot jobs. The workers are controlled by coaster services in that they die after a set idle time if no jobs are scheduled by Swift to run, thus releasing the resource on which they are running. New workers are flexibly spawned when jobs waiting on data get ready to run, for instance. In addition, Coasters helps coordinate data staging (called provider-staging in Swift) and task configuration and packing suitable for the target resources (HPC and HTC). For instance, in this work, Coasters pack 16 tasks per compute node on XSEDE, corresponding to 16 cores on each XSEDE (Ranger) node. Coasters provides services such as building the batch submission scripts to be submitted to the XSEDE compute nodes, probing job-status, staging in the input data, staging out the results, and fault tolerance in the event of network connection failures.

6. THEORETICAL ANALYSIS

In this section, we present a theoretical analysis of the distribution of computation for the CyberShake workflow. Our goal is to reduce time to solution by optimally utilizing opportunistically obtained compute cores and wide-area data bandwidth. Our approach utilizes cores so obtained both from the local HPC resource and from distributed grid resources. While our experiments to date have used OSG as the sole source of the remote resources, an almost identical approach can be applied to leverage cloud resources as well.

Most of the initial segment of the workflow, SGT-generation, must, for all practical purposes, be run on HPC systems, driven by the requirements of the four parallel MPI task stages, PreSGT through SimSGT in Table 1 and Figure 1. (However, task PreCVM could be run on almost any host, such as the interactive or service host on which Swift is typically executed).

In a facility where multiple HPC resources can access a common high-performance shared filesystem, the two parallel instances of the large SimSGT task and the smaller MergeSGT task could be run on any such system. For example, at UChicago, a 17K-core Cray XE with 24-core nodes and a 384-core conventional cluster with 8-core nodes share the same high-performance GPFS filesystem, and the largest jobs could be queued with a bias to the XE while the smaller ones could be queued on the conventional system. Similar configurations exist within XSEDE. Swift can take advantage of this flexibility by replicating and queuing identical jobs to multiple systems, and retaining the first job to start [24].

In contrast to the SGT phase, The postprocessing workflow phase (Figure 2), is dominated by serial tasks that can be run on remote core-scheduled grid resources efficiently. This attribute is most pronounced in the last two stages of the post-processing workflow (*extract* and *seispeak*) which consume about 60% of the total CPU-hour requirement of the workflow. We note that the first three task stages of post-processing depend only in the SCEC sites database and small configuration files, and can be run any time after the workflow starts, well before . We thus focus our attention on the *extract* and *seispeak* tasks stages of the post-processing phase.

For *extract* to run on OSG, we need to move a total of

$$D = 40 \text{ GB} \cdot n + 5.13 \text{ GB} \qquad (1)$$

where n is the number of OSG sites we are executing the job on. From Table 1, this stage will run for 198–794 CPU-hours. If we acquired 2,000 CPUs across OSG, *extract* will finish in $T = 23$ min. To keep 2,000 CPUs utilized, the data movement rate to OSG should be 32 MB/s ($n = 1$) to 583 MB/s ($n = 20$). As the number of OSG sites increases, most of the data movement involves broadcasting the 40 GB dataset.

When we first distributed the post-processing work across OSG, technical problems with our Swift interfaces to OSG's infrastructure prevented our tests from achieving this demand in bandwidth. Hence, in our tests we performed the *extract* computations in XSEDE.

However, our recent tests of GridFTP from the Ranger XSEDE system direct to OSG worker node local filesystems, using GridFTP, indicate that 150 MB/sec is a comfortable average sustainable aggregate transfer rate. Furthermore, recent practical wide-area data transfer speeds have been measured at rates approaching 9 GB/sec, memory-to-memory, between remote OSG storage elements [1].

At such a range of rate, we can move the 40-GB subSGT dataset to, for example, OSG sites with large clusters such as Fermilab, Brookhaven, San Diego, etc – and run multiple replicas of the *extract* task set, creating multiple sites from which subSGT datasets can be sourced to feed the data-intensive *seispeak* stage. We will return to consider this alternative later.

In our analysis of the *seispeak* computations—the last but most CPU-intensive task stage of the CyberShake workflow—we find degrees of freedom which will allow us to optimize the placement of jobs to accelerate the workflow's time to solution. Decisions about the following factors will affect our goal of maximal usable parallelism for this stage:

F1: Location of compute site: OSG or XSEDE

F2: Shared space availability on sites to store job data (see Figure 4)

F3: Spatial and temporal locality of data

F4: Processing time of 4–12 hours on OSG for best throughput as analyzed in §3

F5: Rate of acquisition of free compute cores, that is, the time T_n taken to acquire n compute cores (see Figure 3 and experiment description in §3)

F6: Network bandwidth resulting in a net throughput between host and OSG compute nodes

F7: Number of seispeak jobs n, job cluster size *clust*

F8: Amount of data required per seispeak job $M_{seispeak}$

The following postulates dictate the relationships among the above factors:

P1: The amount of data moved is directly proportional to the number of jobs.

P2: The numbers of freely available cores and the core acquisition rate are nondeterministic and nonlinear functions of time, respectively.

P3: The frequency of data moved is inversely proportional to job cluster size.

P4: The number of jobs are inversely proportional to job size and the job completion time is proportional to job size.

We exploit the above factors to derive formulae that specify a simple model of our analysis and fine-tuned variable values by experiment.

The time required to send n seispeak jobs to OSG workers at network bandwidth D_s between workers and the submit host must be less than or equal to the time T_n required to acquire n cores on OSG. Let $M_seispeak$ and K be the rupture variation and constant subSGT data size. Then, the total memory required by a n jobs of cluster size *clust* will be $n \times clust \times (M_{seispeak} + K)$. The time taken by n seispeak jobs to reach OSG nodes will be given by:

$$\frac{n \cdot clust \cdot (M_{seispeak} + K)}{D_s} \leq T_n \tag{2}$$

Now, let us consider the cluster size from the point of view of best throughput wall-time constraints on OSG. Let, the best throughput wall-time be W_t. Let t_1 be the average execution time taken by one seispeak job. Then, the cluster size is given by

$$clust = \frac{W_t}{t_1} \tag{3}$$

In the case of seispeak jobs, the average job execution time is 7 minutes while the execution tim on OSG for best throughput is 4–12 hours. This means, we need a cluster size of between 35 and 102 or an average of 68.5.

Substituting for clust in Equation 2 from Equation 3:

$$\frac{n \cdot W_t \cdot (M_{seispeak} + K)}{D_s \cdot t_1} \leq T_n \tag{4}$$

This analysis indicates data movement bandwidth is critical to timely availability of data on OSG. Alternatively, various caching strategies could be applied in order to reuse data on a given site for multiple executions. However, the degree of reuse will depend on the nature of application, and caching by nature requires the use of site-wide storage systems as opposed to faster and contention-free node-local filesystems.

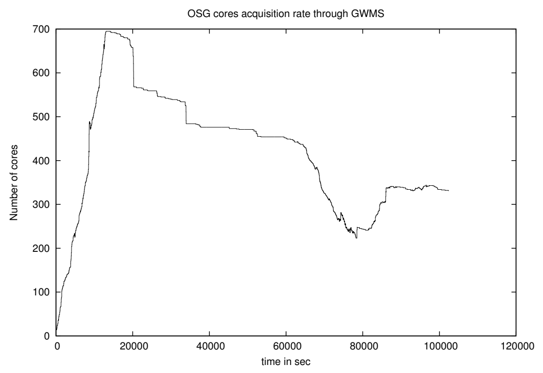

Figure 3: Rate of acquisition of cores on OSG

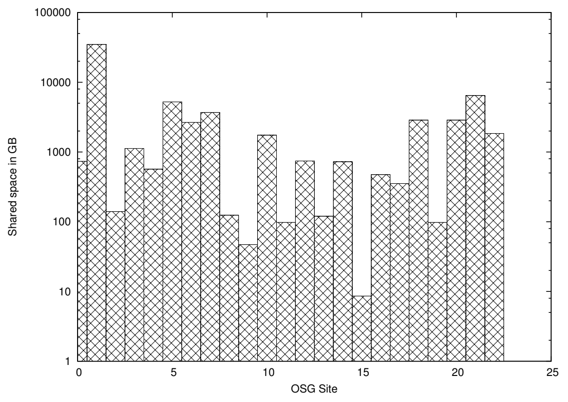

Figure 4: Available shared space on various OSG sites

7. IMPLEMENTATION

We began by simply mapping all of the tasks in the SGT generation phase to XSEDE and all of the tasks in the post-processing phase to OSG. This seemed reasonable, since the SGT generation is dominated by HPC tasks which naturally fit on XSEDE, and the post-processing is all serial tasks which are naturally well-suited OSG.

Our initial implementation of OSG runs was based on data movement using the gridFTP servers. However, the runs were significantly slow because of bottlenecks as a result of a single centralized gridFTP server per site on OSG. Furthermore, several large OSG sites have abstracted their gridFTP servers behind the SRM interfaces making it difficult to access a functioning gridFTP server on those sites. As a workaround to this situation, we moved to pure TCP based data movements coordinated by the coasters directly between submit hosts and OSG worker nodes.

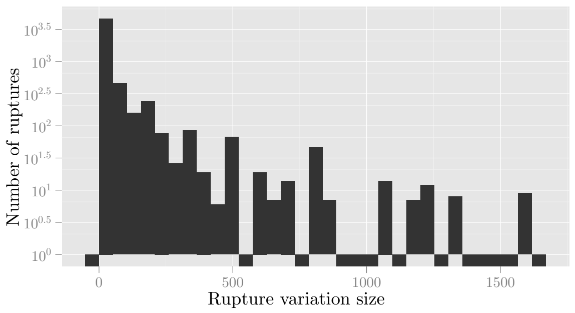

Figure 5: The number of ruptures vs. number of variations per rupture.

Our experiments with both pull and push models of data movement from submit host to OSG sites (see Table 2) show that we have sufficient bandwidth available to sustain a network throughput required for OSG runs.

	netcat(nc)	wget
Best	68 MB/s	104 MB/s
Average	42 MB/s	29 MB/s
Worst	15 MB/s	3 MB/s

Table 2: Data movement times for submit host to push data on a node on wide area network measured using the Linux *netcat (nc)* utility and for workers on various OSG sites to pull data from the submit host using the *wget* utility.

As shown in Figure 5, the majority of ruptures have less than 20 variations. All jobs require moving 164 MB of input. Because of the high data-movement–to–compute ratio for these ruptures, it is best to compute these jobs on XSEDE resources.

The data transfer issues led us to decide not to do all the post-processing on OSG, but to analyze the tasks and only use OSG where it made sense, specifically where the data transfer time for one or more tasks was small compared to the computing time for those tasks.

We started by profiling the workflow as shown in Table 1. We identified the number and nature of tasks, in terms of the model of computation (serial vs. parallel), time and data requirements.

The processing for each variation of a single rupture involves the common use of *extract*, which either requires the SGT data to be moved and *extract* run, or *extract* to be run and the subSGT data to be moved. This means that the compute time needed to process all of the variations of one rupture is proportional to the number of variations of that rupture, and the data transfer time is fixed for that rupture. First, we decided perform the *extract* tasks on XSEDE, because the ratio of input to output data is quite high, and the compute time for these tasks is only a few minutes. Next, analyzing the data transfer and compute times led us to perform *seispeak* tasks on XSEDE for ruptures with less than 20 variations, as this was roughly the crossover point between data transfer time and compute time for these tasks.

An additional consideration is scheduling time, as we want to create OSG tasks of at least 4 hours (See §3). This led us to divide the work for ruptures with more than 60 variations into groups of at most 60 variations, since each group would have about 4 hours computing, which is a good balance between 1) wanting to increase this time to increase system utilization and to increase the compute-transfer ratio and 2) wanting to decrease this time to create good load balancing and avoid trailing task problems [2]. The clustering is shown in Figure 6.

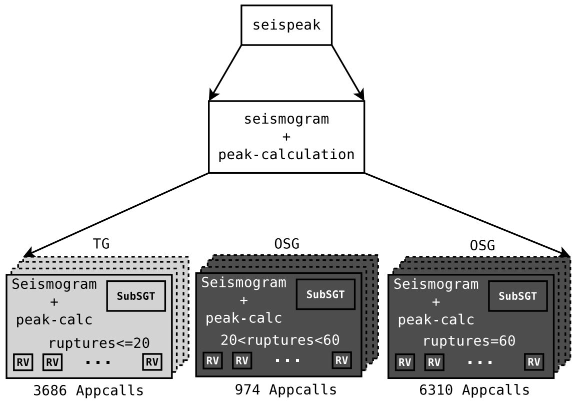

Figure 6: Expanded representation of seispeak jobs detailing their data requirements, cluster sizes and distribution over XSEDE and OSG

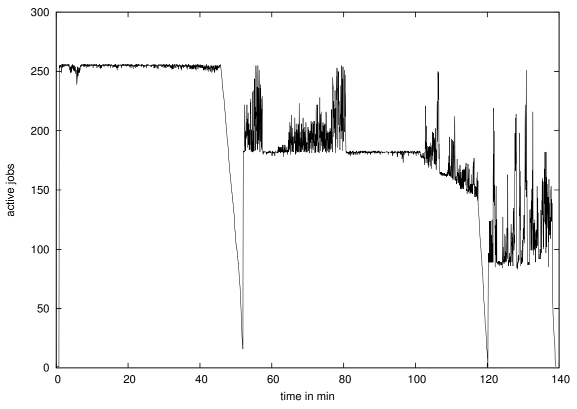

Figure 7: Activity plot showing the number of jobs active in parallel over time running on XSEDE compute nodes. A peak activity of 256 jobs was obtained on XSEDE resources completing 21,000 jobs in less than 3 hours.

A sample of about 5% of the total clustered *seispeak* tasks running on XSEDE compute nodes is shown in Figure 7, with 21,000 jobs completing in less than 3 hours.

The time-to-solution goal of 1–2 days for the 11-stage pipeline requires a continuous run over this period. During a run of this length on a distributed infrastructure, there will be many intended and unintended conditions that might cause system crashes or connection breakage, e.g. system reboot or planned maintenance. The *resume* feature of Swift has been handy in such situations. Swift maintains a running inventory of tasks completed including the links to associated data. In the event of a crash, the resume feature enables the workflow to start from the point it left at the time of crash.

In addition, tasks can fail. One cause is that the compute nodes of the Ranger XSEDE resource have a fixed amount of node-wide shared memory, and packing the maximum number of post-processing tasks into one node can cause the node to exceed its memory limit. This causes approximately 5% of tasks to fail in some of our tests. It would be possible (but complex) to predict this situation and avoid it by changing the mapping of tasks to nodes or limiting the number of tasks per node. Here, because this situation does not occur often, we simply use the *retry* feature of Swift, where a failed task is retried a set number of times before it is treated as a failed task.

8. CONCLUSIONS AND FUTURE WORK

We show characterization, automation, and generalization of the CyberShake application, which has complex and challenging data requirements. We use Swift to distribute and execute the tasks, move the data, and handle fault conditions. With a carefully planned workflow task-level decomposition, we achieve a high sustained and balanced load on both the XSEDE and OSG infrastructures.

This work demonstrates a planned distribution of computation and data movement can transparently and seamlessly achieve integration among heterogeneous code, diverse data requirements and disparate infrastructures while giving a desired performance level.

Today's scientific applications are increasingly dominated by high data requirements. Consequently, planning the workflow around the data movement becomes a critical part of any execution: solutions must take into account properties of data and cost of its movement.

Furthermore, it is not trivial to address the diversity in the computational characteristics found within a single application. It required significant efforts in development and enhancement of the Swift coasters framework in order to address a targeted execution of parallel and serial code on infrastructures. We believe this development will cater the needs of other similar applications.

We see collective data management (CDM) [25] as a promising technique to run applications with characteristics similar to those of ours. Work is underway on enhancement in CDM policies resulting in a pre-distribution of data on OSG sites similar to scatter-gather paradigm. We anticipate this will benefit the applications by advanced scattering of the required data to the shared disk of OSG sites.

Acknowledgment

This research was funded in part by NSF under ExTENCI (grant OCI-1007115), by NSF through TeraGrid resources provided by TACC under grant TG-ASC090068, and by the U.S. Department of Energy under contract DE-AC02-06CH11357. It also used Open Science Grid resources supported by diverse sources, including DOE and NSF.

9. ADDITIONAL AUTHORS

10. REFERENCES

[1] Advanced Network and Distrbuted Storage Laboratory website.

[2] T. G. Armstrong, Z. Zhang, D. S. Katz, M. Wilde, and I. Foster. Scheduling many-task workloads on supercomputers: Dealing with trailing tasks. In *Proceedings of Many-Task Computing on Grids and Supercomputers, 2010*, 2010.

[3] P. Avery, R. Roskies, and D. S. Katz. ExTENCI: Extending Science Through Enhanced National Cyberinfrastructure, 2010. Project homepage: `https://sites.google.com/site/extenci/`.

[4] S. Callaghan, E. Deelman, D. Gunter, G. Juve, P. Maechling, C. Brooks, K. Vahi, K. Milner, R. Graves, E. Field, D. Okaya, and T. Jordan. Scaling up workflow-based applications. *Journal of Computer and System Sciences*, 76(6):18, 2010.

[5] S. Callaghan, P. Maechling, E. Deelman, K. Vahi, G. Mehta, G. Juve, K. Milner, R. Graves, E. Field, D. Okaya, D. Gunter, K. Beattie, and T. Jordan. Reducing Time-to-Solution Using Distributed High-Throughput Mega-Workflows - Experiences from SCEC CyberShake. In *Fourth International Conference on eScience*, pages 151–158, 2008.

[6] P. Couvares, T. Kosar, A. Roy, J. Weber, and K. Wenger. *Workflow Management in Condor*, pages 357–375. Springer, 2007.

[7] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M.-H. Su, K. Vahi, and M. Livny. *Pegasus: Mapping Scientific Workflows onto the Grid*, volume 3165, pages 131–140. Springer Berlin / Heidelberg, 2004.

[8] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[9] A. Espinosa. Cybershake on Opportunistic Cyberinfrastructures. Master thesis, University of Chicago, Chicago, Mar. 2011.

[10] A. Espinosa, D. S. Katz, M. Wilde, K. Maheshwari, I. Foster, S. Callaghan, and P. Maechling. Data-intensive CyberShake computations on an opportunistic cyberinfrastructure. In *Proceedings of the 2011 TeraGrid Conference: Extreme Digital Discovery*. ACM, 2011.

[11] I. Foster and C. Kesselman. *The Globus toolkit*, pages 259–278. Morgan Kaufmann Publishers Inc., 1999.

[12] R. Graves, T. Jordan, S. Callaghan, E. Deelman, E. Field, G. Juve, C. Kesselman, P. Maechling, G. Mehta, K. Milner, D. Okaya, P. Small, and K. Vahi. CyberShake: A Physics-Based Seismic Hazard Model for Southern California. *Pure and Applied Geophysics*, Online Fir:1–15, May 2010.

[13] M. Hategan, J. Wozniak, and K. Maheshwari. Coasters: uniform resource provisioning and access for clouds and grids. In *4th IEEE/ACM International Conference on Utility and Cloud Computing (UCC 2011)*, Dec. 2011.

[14] D. S. Katz, S. Callaghan, R. Harkness, S. Jha, K. Kurowski, S. Manos, S. Pamidighantam, M. Pierce, B. Plale, C. Song, and J. Towns. Science on the TeraGrid. *Computational Methods in Science and Technology*, Special Issue 2010:81–97, 2010.

[15] P. Maechling, E. Deelman, L. Zhao, R. Graves, G. Mehta, N. Gupta, J. Mehringer, C. Kesselman,

S. Callaghan, D. Okaya, H. Francoeur, V. Gupta, Y. Cui, K. Vahi, T. Jordan, and E. Field. *SCEC CyberShake Workflows – Automating Probabilistic Seismic Hazard Analysis Calculations* , pages 143–163. Springer London, London, 2007.

[16] J. McGee and C. Sehgal, 2011. Personal communication.

[17] R. Pordes, D. Petravick, B. Kramer, D. Olson, M. Livny, A. Roy, P. Avery, K. Blackburn, T. Wenaus, F. Würthwein, I. Foster, R. Gardner, M. Wilde, A. Blatecky, J. McGee, and R. Quick. The open science grid. *Journal of Physics: Conference Series*, 78:012057, July 2007.

[18] I. Raicu, Y. Zhao, I. T. Foster, and A. Szalay. Accelerating large-scale data exploration through data diffusion. In *Proceedings of the 2008 International Workshop on Data-Aware Distributed Computing (DADC '08)*, pages 9–18. ACM Press, June 2008.

[19] K. Ranganathan and I. Foster. Simulation studies of computation and data scheduling algorithms for data grids. *Journal of Grid Computing*, 1(1):53–62, 2003.

[20] D. Reed. Grids, the TeraGrid and beyond. *Computer*, 36(1):62–68, Jan. 2003.

[21] M. Rynge, G. Juve, G. Mehta, E. Deelman, K. Larson, B. Holzman, I. Sfiligoi, F. Würthwein, G. B. Berriman, and S. Callaghan. Experiences Using GlideinWMS and the Corral Frontend Across Cyberinfrastructures. In *Proceedings of the 7th IEEE International Conference on e-Science (e-Science 2011)*, 2011.

[22] I. Sfiligoi, D. Bradley, B. Holzman, P. Mhashilkar, S. Padhi, and F. Würthwein. The Pilot Way to Grid Resources Using glideinWMS. In *Computer Science and Information Engineering, 2009 WRI World Congress on*, pages 428–432, 2009.

[23] G. von Laszewski, I. Foster, J. Gawor, and P. Lane. A Java Commodity Grid Kit. *Concurrency and Computation: Practice and Experience*, 13(8-9), 2001.

[24] M. Wilde, M. Hategan, J. M. Wozniak, B. Clifford, D. S. Katz, and I. Foster. Swift: A language for distributed parallel scripting. *Parallel Computing*, 37(9):633–652, 2011.

[25] J. M. Wozniak and M. Wilde. Case studies in storage access by loosely coupled petascale applications. In *Proc. 4th Annual Workshop on Petascale Data Storage*, pages 16–20, 2009.

[26] XSEDE Project. XSEDE web site.

[27] L. Zhao, P. Chen, and T. Jordan. Strain Green's tensors, reciprocity, and their applications to seismic source and structure studies. *Bulletin of the Seismological Society of America*, 96(5):1753–1765, 2006.