

# Cybershake on opportunistic cyberinfrastructures

Allan M. Espinosa

Friday 4<sup>th</sup> March, 2011

### **Abstract**

Continued improvements in provisioning capabilities of cyberinfrastructures increase the computational enablement of science such as the Cybershake application. Using workflow tools such as Swift and Pegasus, we observed how jobs are scheduled to exploit an expanded set of resources available in TeraGrid and Open Science Grid (OSG). Our tests on OSG gave 2000 CPUs in a day on an opportunistic basis. This was well within the time-to-solution requirements of the Cybershake application. However, there were potential bottlenecks in the data requirements of the application. In this paper, we characterize how the time-to-solution budget is spent by looking at the running time and file transfer time of the post-processing computation.

# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Cybershake . . . . .	5
1.2	Opportunistic resource usage . . . . .	6
<b>2</b>	<b>Cyberinfrastructures</b>	<b>8</b>
2.1	TeraGrid . . . . .	8
2.2	Open Science Grid . . . . .	9
<b>3</b>	<b>Cybershake computation</b>	<b>10</b>
3.1	Earthquake Rupture Forecast . . . . .	10
3.2	SGT . . . . .	11
3.3	Post-processing . . . . .	11
<b>4</b>	<b>Related work</b>	<b>14</b>
4.1	Pegasus WMS . . . . .	14
4.2	Cybershake Production . . . . .	15
4.3	Pilot jobs . . . . .	15
<b>5</b>	<b>Implementation</b>	<b>17</b>
5.1	Swift . . . . .	17
5.2	Data . . . . .	18
5.3	Workflow . . . . .	19
5.4	Pilot jobs . . . . .	20
5.5	Globus Online . . . . .	22
<b>6</b>	<b>Performance model</b>	<b>23</b>
6.1	Compute component . . . . .	23
6.2	Opportunistic potential . . . . .	25
6.3	Data component . . . . .	27
6.3.1	<i>extract()</i> . . . . .	27
6.3.2	<i>seispeak()</i> . . . . .	30
6.4	Transfer measurements . . . . .	31
6.5	Data-specific plan . . . . .	32
6.6	Alternative workflow . . . . .	35

<b>7 Conclusion</b>	<b>37</b>
<b>8 Future work</b>	<b>38</b>
<b>9 Acknowledgements</b>	<b>39</b>

# List of Figures

1.1	Hazard curve of Whittier Narrow near Los Angeles from [1]. . . . .	6
1.2	Hazard map of Southern California from [1]. . . . .	7
3.1	The post-processing workflow to generate a hazard curve in Laguna Peak, Los Angeles. Note that the numbers presented in this figure refer to a sample run described in Chapter 6. . . . .	12
5.1	Rupture variation sizes for the site <i>LGU</i> . . . . .	20
5.2	Schematic of a manual worker submission engine to a persistent coaster service. . . . .	21
6.1	Runtime distribution of <i>extract()</i> jobs in PADS. The average runtime is 73.35 s. . . . .	24
6.2	Runtime distribution of <i>seismogram()</i> from a sample of 22,690 jobs out of 405,378. The average runtime is 157.78 s. . . . .	24
6.3	Peak ground acceleration job distribution from a sample of 21,956 jobs out of 405,378. The average runtime is 17.34 s. . . . .	25
6.4	Four hour worker jobs requested on 18 OSG computing resources. . . . .	26
6.5	Data size distribution of rupture variation files needed by <i>extract()</i> jobs. The average size is 864.14 kB. . . . .	28
6.6	Data size distribution of subSGT meshes. The average size is 163.43 MB. . . . .	28
6.7	Data distribution of rupture variations needed by <i>seispeak()</i> jobs. The average size is 4.09 MB. . . . .	30
6.8	Datarate of sending rupture variation files to 15 OSG resources from PADS. . . . .	33
6.9	Performance of transferring subSGT meshes together with rupture variations. The total data sent over this period is 424.52 GB. . . . .	35

# List of Tables

3.1	Average number of jobs per geographic site in the Cybershake database. . . . .	13
6.1	Transfer workload of 10,000 rupture variation files. Data is sent from PADS to 15 OSG resources. . . . .	33
6.2	Transfer workload of 10,000 rupture variation files and subSGT meshes. Data is sent from PADS to 15 OSG resource. . . . .	34

# Chapter 1

## Introduction

Probabilistic seismic hazard analysis (PSHA) are now used for making predictive earthquake forecasts [2]. Ground motions are estimated for a geographic site by quantifying all possible earthquakes that can affect it. The result is presented as a hazard curve. The curve shows the probability of exceeding ground motion levels like in Figure 1.1. This information is used as a basis for building codes in the site's area [1]. A set of hazard curves from multiple sites can be integrated together to form a regional hazard map. The map represents the ground motion level for an area at a given probability like in Figure 1.2.

PSHA is computed from two inputs: first is a list of possible earthquakes that might affect a site called ruptures. Such a list is obtained from an Earthquake Rupture Forecast (ERF). The second input is a method to calculate ground motions produced from each earthquake. Empirical attenuation relationships are used to generate these motions. However, this technique do not cover the full range of possible earthquake magnitudes [3]. The Southern California Earthquake Center (SCEC) uses 3D ground motion simulations with analestic wave propagation models to address this limitation [1, 4]. Variability in the earthquake process is taken account into the SCEC's model. We will refer to these variabilities as rupture variations.

### 1.1 Cybershake

SCEC implements this physics-based model through their Cybershake platform. Cybershake's hazard curve generation process is compute and data intensive. For a single geographic site, it takes 17,000 CPU-hours to generate its hazard curve. Data produced throughout this computation totals to 750 GB. The tasks involved to complete the curve involves tightly-coupled jobs (SGT generation) and embarrassingly parallel jobs (post-processing). The current production system computes curves within a 24 to 48 hour time-to-solution on a single NSF TeraGrid resource. Runs were made on 800 CPU reservations. The speedup time obtained is  $O(1000)$  [1]. Details of the entire computation is found in

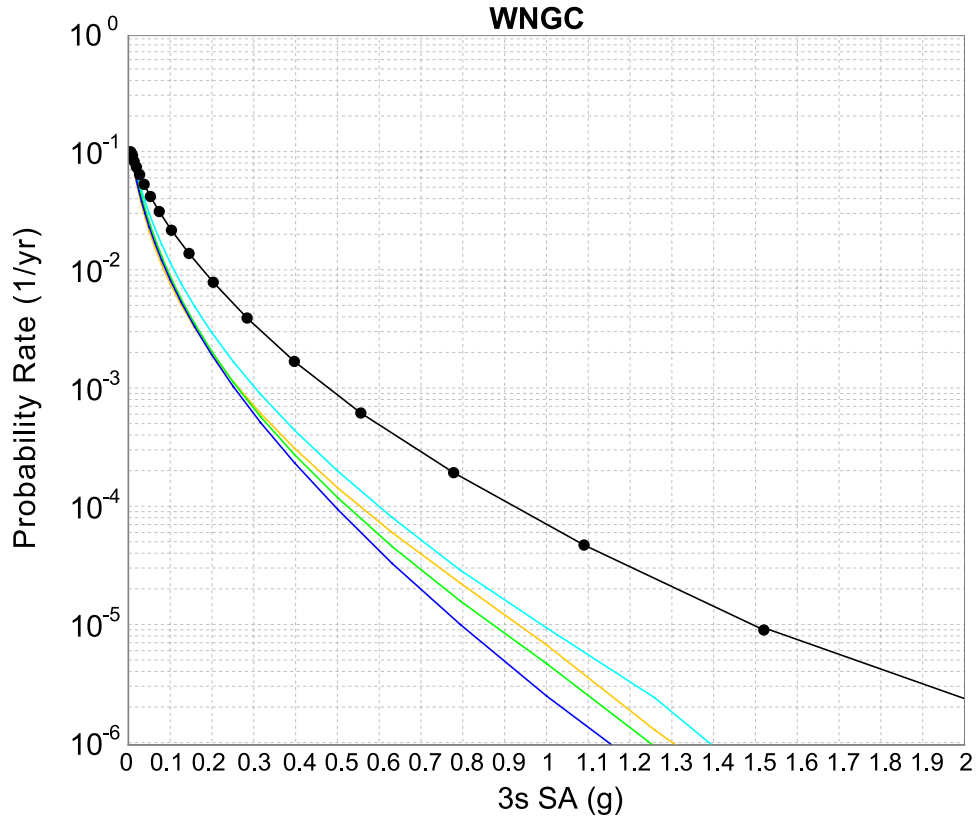


Figure 1.1: Hazard curve of Whittier Narrow near Los Angeles from [1].

Chapter 3.

To date, SCEC runs the Cybershake platform creates hazard curve computations at around 250 sites in Southern California as shown in Figure 1.2. Their goal is to generate more hazard curve at a finer spatial resolution of 5 km in the region. This translates to around 2,000–10,000 geographic sites.

## 1.2 Opportunistic resource usage

Work in this paper is mainly motivated by the Extending Science Through Enhanced National Cyberinfrastructure (ExTENCI) project [5]. The project aims to expand the compute resources of SCEC by using CPUs available in the Open Science Grid (OSG) [6] in addition to TeraGrid [7]. We observed that OSG opportunistically offers around 2000 CPUs in a day. However, these cycles are spread over 25 compute elements. Our initial runs suffered significantly from the data transfer involved to prepare 415,000 jobs. In this paper, we identify the bottlenecks involved in performing the post-processing jobs. Two imple-

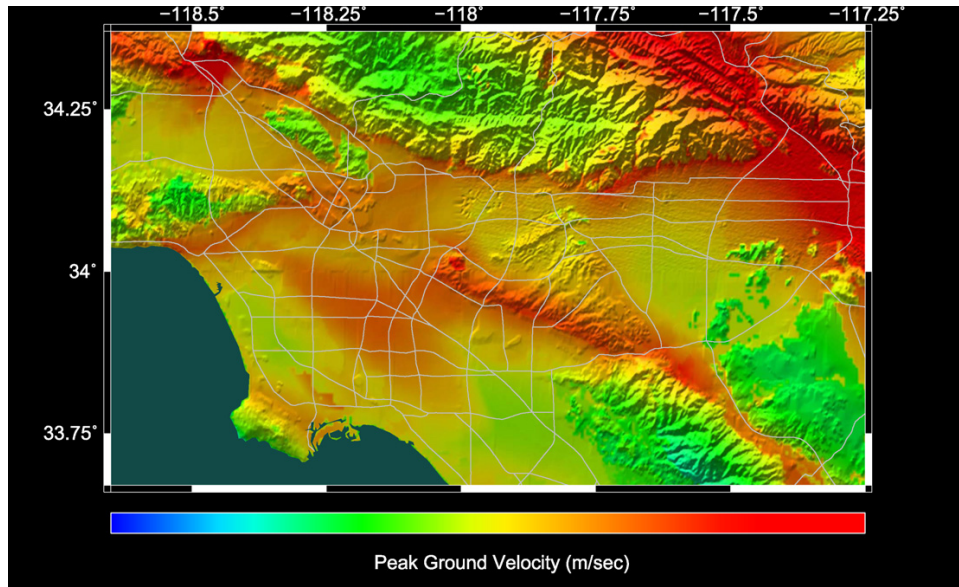


Figure 1.2: Hazard map of Southern California from [1].

mentations of the workflow were studied. The first is the existing Cybershake implementation in Pegasus [8]. We also reimplemented the post-processing component in Swift [9] to see how the workflow behaves in an adaptive scheduler in contrast to Pegasus' pseudodynamic planning. Chapter 4 discusses in detail this plan generation process on multiple compute sites. Then in Chapter 5, we observed in our Swift implementation that staging input files to OSG resources is slow. Some models in Chapter 6 are proposed and evaluated to improve the time-to-solution budget of Cybershake.

## Chapter 2

# Cyberinfrastructures

Computing technologies improve modes of scientific discovery [10, 11]. It enables large scale simulation of phenomena that cannot be easily reproduced by experiments. However, component-wise improvements such as increase in microprocessor speeds, wider network systems and higher data storage platforms is not enough to accelerate scientific breakthrough. Scientific software, data management platforms, and enabling frameworks are also essential. The National Science Foundation (NSF) refers to the infrastructure needed to fully utilize such information technology improvements as *cyberinfrastructures* (CI) [12].

A CI integrates hardware resources such as computing facilities, data storage systems and collaboration networks through platform independent software tools for inter-operation. It creates interdisciplinary teams with domain science experts and software engineers to develop applications that exploits the full potential of cyberinfrastructures. The Cybershake computation platform generates 50-year projections worth of probabilistic seismic curves by managing the complexities of running the simulations on high performance computing resources available like TeraGrid. In this chapter, we describe the CI facilities that are used to run Cybershake.

### 2.1 TeraGrid

TeraGrid is a research computing infrastructure that combines large computing and data management facilities [7]. Completed in 2003, it interconnected the National Center for Supercomputing Applications (NCSA), San Diego Supercomputer Center (SDSC), University of Chicago/Argonne National Laboratory Mathematics and Computer Science Division (UC/ANL-MCS), and California Institute of Technology Center for Advanced Computing Research (CalTech). Additional institutions that are part of the infrastructure today are Pittsburgh Supercomputing Center (PSC), Indiana University (IU), Louisiana Optical Network Initiative (LONI), National Center for Atmospheric Research (NCAR), National Institute for Computational Science (NICS), Oak Ridge National Lab-

oratory (ORNL), Purdue University, and Texas Advanced Computing Center (TACC). From its initial capability of 24 Tflops of computing power and 800 TB of fast disk storage, TeraGrid now has 2 Pflops and 50 PB worth of capacity [13]. The facilities in TeraGrid are linked through a 10 Gbit/s optical network.

The software stack in the TeraGrid sites are integrated with open source software. Most sites run Linux for its operating system. The software stack typically include a wide range of compilers like GCC, Intel, and Portland Group. Various implementations of the Message Passing Interface are also included on top of these compilers. Cluster management tools for submitting and managing jobs are varied. Sites have either PBS Torque/Maui, Sun Grid Engine (SGE), Condor, or Cobalt installed. For the sites to communicate, all have an installation of the Globus Toolkit [14]. Globus provides software for inter-site communications such as remote resource execution, data transfer and secure authentication.

Cybershake currently runs in production in TeraGrid sites like NCSA Abe, and TACC Ranger [1]. For the experiments in this paper, we ran the Cybershake SGT and portions of post-processing workflows in TACC Ranger.

## 2.2 Open Science Grid

The Open Science Grid (OSG) is a consortium that links distributed grid clusters that [6]. This cyberinfrastructure mainly supports the data handling analysis of several large physics experiments such as the LHC, LIGO, STAR, etc. In addition, it enables other science groups access to these resources via opportunistic use that are owned by other institutions. The platform makes it easy to add their own resources to OSG's common middleware stack, the Virtual Data Toolkit (VDT). Participants of the consortium include science communities, DOE laboratories, DOE/NSF university computing, networking and storage facilities and groups that provide the middleware [6]. Currently, there are 80 consortium members [15].

All the test runs in this paper were conducted under the Engagement virtual organization. The OSG Engagement team provides support for scientists who are new to accessing cyberinfrastructures. Its Resource Selection Service (ReSS) monitors OSG and provides a guide for increasing opportunistic usage of computing resources [16]. Currently, ReSS reports 30,000 to 40,000 CPUs that can be opportunistically tap. Chapter 6 describes our experiments to determine the average number of CPUs we can tap from OSG.

## Chapter 3

# Cybershake computation

PSHA characterizes earthquake hazards by quantifying peak ground motions from all possible earthquakes that might affect a particular site. It computes the probability of a site experiencing ground motion over a time period. This chapter describes how the computation is organized in terms of computing hardware, application software and middleware to produce a seismic hazard curve. This plot is used by civil engineers to estimate seismic hazards of a planned building construction. PSHA curves are also used as basis for regional building codes [2]. A set of curves describing many sites can be combined to generate a hazard map of a larger geographical region.

### 3.1 Earthquake Rupture Forecast

There are two primary computation elements in generating PSHA curves [17]. The first component is the Earthquake Rupture Forecast (ERF). ERF is a program that produces a list of earthquakes that may occur in a region. For the Cybershake runs in SCEC, the region of Southern California is used for production runs and tests. ERF describes earthquakes in terms of its magnitude, affected fault surface and probability to occur. This list is generated from faults in a region described by its activity, size, historical earthquake record, slip rates, and other geological and geophysical information [2,18]. We describe each item in this list as a *rupture*.

The next component is an Intensity Measure Relationship (IMR). It describes earthquake wave decay as a function of distance. At a particular geographic site of interest, IMR shows the ground level motion produced by an earthquake. The Cybershake computation platform generates replaces traditional attenuation-based IMR with waveform simulation-based ones. This new method of generating curves have scientific advantages by improving the quality of available seismic hazard information [1]. But the ruptures in the ERF does not contain all earthquakes that must be simulated. The new model requires information on how individual earthquake rupture occurs. All possible variations

are simulated by generating hypocenter and slip distributions for each earthquake rupture [2]. We refer to the data describing the variations as *rupture variations*.

The rupture variations dataset in the ERF component for production Cybershake runs is only updated infrequently. For typical SCEC runs, rupture variations are considered as data input since it only gets updated once a year. The ERF dataset we used in this paper consists of 14,856 ruptures. Each rupture has an average 44.72 rupture with average size 145.63 MB. The total rupture variations for the ERF dataset is 664,432 rupture variations with total size of 2.16 TB.

IMR is the core computation component of the Cybershake platform to generate a 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. Second, SGT components are extracted from each relevant earthquake rupture. Next, synthetic seismograms are generated for each rupture to post-process the SGT. Finally, peak spectral acceleration values are obtained from the seismograms. These values are combined to form the hazard curve [3].

## 3.2 SGT

Given geographic site of interest, a resulting SGT is needed. The data consists of approximately 1.5 billion mesh points with seismic wave velocity information [3]. A tightly-coupled program implemented with the Message Passing Interface (MPI) is used to create this dataset. The code uses 400 CPUs for creating the  $x$  and  $y$  components of the mesh. Each run will last for 14 hours consuming 10,000 CPU-hours in total. The  $x$  and  $y$  components are saved in individual files of 20 GB each giving a total of 40 GB SGT data for each geographic site.

Current Cybershake production runs perform the SGT mesh generation on various TeraGrid computing resources such as NCSA Abe, and TACC Ranger [1,19]. In addition, computing cluster resources from SCEC/USC are also used in complement to the dedicated computing centers. For the experiments in this paper, we treated the SGT component of the computation as input data coming in from TeraGrid resources.

## 3.3 Post-processing

The last three steps of the hazard curve generation process are loosely-coupled in nature. The Cybershake platform groups these components that required high-throughput computation into a separate execution environment or workflow. Data and computation requirements in this section are obtained from prior SCEC work on Cybershake. The numbers used for the models in Chapter 6 were obtained by sampling an actual run. Figure 3.1 shows an overview of the entire post-processing computation of this run.

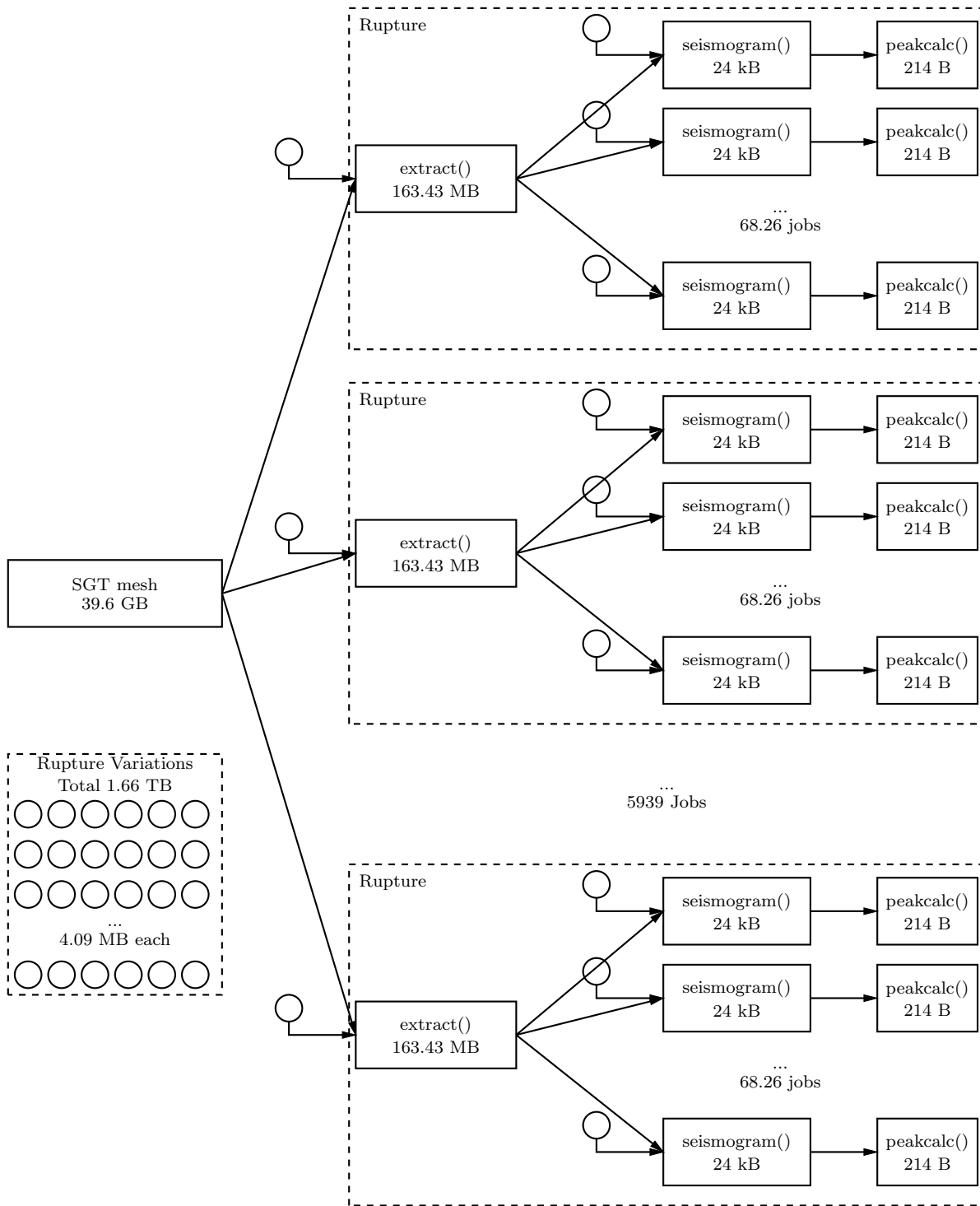


Figure 3.1: The post-processing workflow to generate a hazard curve in Laguna Peak, Los Angeles. Note that the numbers presented in this figure refer to a sample run described in Chapter 6.

Table 3.1: Average number of jobs per geographic site in the Cybershake database.

Job	Number of Jobs	
	Mean	Standard Deviation
<i>extract()</i>	6,889	501
<i>seismogram()</i>	419,195	21,985
<i>peak_calc()</i>	419,195	21,985

Subcomponents of the strain Green’s tensors are first obtained to prepare for the core of the post-processing computation. Each subcomponent of the SGT corresponds to an earthquake rupture within a 200 km radius of the geographic site of interest [17]. On average, a geographic site can be affected by 7,000 earthquakes. This corresponds to 7,000 jobs in the post-processing workflow. We refer to this jobs that generates SGT subcomponents as *extract()* jobs. The data requirements of this job is as follows: for input, the  $x$  and  $y$  components of the SGT mesh (40 GB) and earthquake rupture information from the ERF (2.36MB). For output, the subcomponents (159.12 MB) of the SGT mesh corresponding to a rupture. The total computation time for all the *extract()* jobs is 250 CPU-hours [1].

The next step is seismogram synthesis using seismic reciprocity [4]. This task is performed for each rupture variation associated with an SGT subcomponent. Runtimes of each seismogram synthesis task is in the order of 1-60 minutes as this computation technique is inexpensive to calculate. For the 7,000 earthquakes, there is a total of 415,000 rupture variations that are used to generated the same number of seismograms. This makes the post-processing component of the hazard curve generation data-intensive. The input required for each *seismogram()* task is the  $x$  and  $y$  subcomponents of the SGT for the rupture (159.12 MB) and one of the rupture variations in the associated earthquake (2.36 MB). Each of the 415,000 seismograms produced has a size 24 kB producing a total of 10 GB worth of output.

Lastly, peak spectral acceleration at 3.0 seconds is obtained from the seismograms to generate the intensity measures needed for the hazard curve. We refer to this component of the computation as *peak\_calc()* jobs. Taking in each of the seismogram generated from the previous step it will produce a total output of 90 MB. Each of the intensity measure data file generated has a size of 214 B.

Aside from the computation and job requirements found in literature, we also measured statistics from the Cybershake database of production runs. Based our database queries, Table 3.1 shows the average number of jobs per component of the post-processing workflow. There were 918 geographic sites in the database when these statistics were sampled. Sample runs for the geographic site *LGU* were also run. We describe the runtime distribution of this instance of the workflow in Section 6.1.

# Chapter 4

## Related work

Most applications that run on cyberinfrastructures are composed from existing application components. In the Cybershake post-processing computation described in Chapter 3, the application is described through a workflow of three different serial programs: *extract()*, *seismogram()* and *peak\_calc()*. The current production code of Cybershake currently uses the Pegasus planning system to generate its workflows [3].

### 4.1 Pegasus WMS

The post-processing computation in Cybershake is described as an abstract workflow. Each step only describes the logical data that is produced and consumed. Details on how a program is executed in a computing resource is abstracted away. The portability of the workflow allows it to be executed on a variety of resources. The Pegasus Workload Management System (WMS) manages the transfer and computation tasks of an abstract workflow [8]. The abstract workflow of Pegasus is expressed as a direct acyclic graph written in XML. Pegasus WMS calls this component a “DAX”. In Cybershake, the DAX is generated through a Java program with the geographic site as the input.

The core engine of Pegasus that determines which resources it executes the application is the mapper. In addition to the abstract workflow expressed as a DAX, the mapper requires a transformation catalog and a resource definition. The transformation catalog contains the list of resources that can run a component of the computation in the workflow. After computation is mapped to available resources, a “concrete workflow” is produced as a DAGman workflow [20].

Another component in the mapping process is the replica catalog. As data elements are produced in the computation, Pegasus adds replica registration jobs. These tasks gives the mapping of the logical name of files described in the abstract workflow to physical filenames found in distributed storage resources. For production work, Pegasus WMS uses the Globus Replica Location Service

to find data in the planning process.

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* [8]. 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.

## 4.2 Cybershake Production

The SCEC team reported scaling issues in the mapping process of the abstract workflow due to the volume of jobs being produced. Recursive DAGs [20] 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 Glideins [1, 21].

Job clustering was also performed to reduce the number of jobs being sent to the Condor pool handling the Glideins [3].

## 4.3 Pilot jobs

Pilot jobs are composed of two components: a user queue, pilot factory. The user queue provides a single-view for heterogeneous resources [21]. The next component in these systems is a “pilot factory”. When a user requests a job to the user queue, the pilot factory brokers a pilot job request to prospective cyberinfrastructure resources. Once the request is approved, the pilot job starts to get work from the user queue. Example implementations of pilot job systems include GlideinWMS [22], MyCluster [23].

Pilot jobs can also be used as a resource container to implement multi-level scheduling policies [24]. This scheme improves throughput by reducing scheduling overhead of short-duration jobs [25]. A longer-duration pilot job requested in a remote resource can be reused to run multiple shorter-duration user tasks. Initial problems on using existing schedulers like Condor, PBS, and SGE for managing these kinds of jobs in user queues creates scalability problems because of the volume of jobs received. Systems that use existing schedulers like GlideinWMS solves this problem by creating a load-balanced cluster of user queues [22]. Falkon [24] and Swift Coasters [26] uses an alternative approach by implementing a light-weight scheduler that can execute user jobs at a higher rate. This removes the need for workflows with high volumes of short-duration jobs to reduce its job request rate to the pilot scheduler.

Swift’s Coaster system has two components: the coaster service and workers. The coaster service receives job requests from the client running the workflow. By default, this service is deployed on the head node of the remote resource.

Then it performs lightweight scheduling to dispatch these jobs to the workers. The workers execute the received jobs on the compute node. Workers are requested to remote resources as jobs whose execution time is several factors larger than the jobs requested by the client workflow. This factor is computed using job allocation configurations in the Swift's site catalog [26].

## Chapter 5

# Implementation

Most of the experiments in redesigning the Cybershake platform to run on OSG is implemented using the Swift workflow engine. Our implementation in this section and analysis in Chapter 6, deals with computing the hazard curve in the geographic site Laguna Peak near Los Angeles. We shall refer to this site as *LGU* throughout the paper.

### 5.1 Swift

Scientific applications are usually patterned as an execution of a set of tasks that are coupled in terms of passing an output of an application as input to the next. “Loosely coupled” computing issues that arise in this computation pattern include management of datasets and tasks in large numbers. Swift handles these problems using its scripting language specification [9]. It uses the XML Data set Typing and Mapping (XDTM) [27] to create a mapping between logical structures in the workflow and physical datasets such as files and GridFTP endpoints. In addition, programming constructs are added on top of the CoG Karajan execution engine [28] to support procedures that operate on logical structures defined in XDTM. The next two sections describes the Cybershake platform based on its logical mapping and operational constructs.

Swift’s runtime environment provides a scalable scheduling of tasks and data transfers in a Swift script program. CoG Karajan’s light-weight threading methods are used to dispatch tasks and can run large execution graphs as previously Swift’s early evaluation results [9]. In contrast to planning systems like Pegasus [8], the assignment of a task’s execution to a remote resource is determined at runtime. Swift exploits this feature by using a feedback mechanism to influence its scheduling decisions. This provides a reliability system that can detect transient problems in remote resources. For example, if a remote computing site *A* suddenly goes down while executing job *X*, Swift will reschedule the job to different site *B*. The opportunistic scheduling algorithm observes the performance of a remote site and sends more jobs to “well-performing” resources [29].

Our workflow implementation runs on both TeraGrid and OSG resources. Assume that in the beginning, 80% of the computation is performed on TeraGrid. Then at a particular point, OSG becomes free and started to offer more CPUs. Swift will receive feedback that OSG is starting to perform well and starts to send more jobs to OSG. This allows TeraGrid and OSG to work together in computing Cybershake.

## 5.2 Data

Our workflow organizes Cybershake data per type of computation. The first part of the computation are the *extract()* jobs. It retrieves a portion of the geographic site's SGT to its corresponding rupture. The necessary data for input are the geographic site, SGT and rupture. A geographic site holds information such as its name and geophysical location in terms of latitude and longitude. Additional information on what type of ERF to use is also identified in this data structure.

```
type Station {
    string name;
    float lat;
    float lon;
    int erf;
    int variation_scenario;
}
```

An `Sgt` dataset physically corresponds to two files representing the  $x$  and  $y$  components of the mesh. For the runs conducted in this paper, the SGT corresponds to two files named “LGU\_fx.sgt” and “LGU\_fy.sgt”. This is also the data definition of SGT components called `subsgt`.

```
type SgtDim;
type Sgt {
    SgtDim x;
    SgtDim y;
}
```

A Rupture is described in terms of three integers according to the Cybershake database. These parameters are used to query the list of variations described as file objects. For this run, there are 5,939 ruptures.

```
type Rupture {
    int source;
    int index;
    int size;
}
```

The other data types like `SgtDim`, `Variation` (rupture variation), `Seismogram` and `PeakValue` are simple file objects.

## 5.3 Workflow

This section describes the post-processing workflow implemented in Swift. First we describe the *LGU* SGT data and its site information using the XDTM definitions in the previous section.

```
Site site = "LGU";
Sgt sgt <"Location of data produced from the
        SGT workflow in TeraGrid">;
```

With the site information, the Cybershake database is then queried to get the list of ruptures. This represents all the possible earthquakes that can affect this geographic location.

```
Rupture rups[] = get_rupture(site);
```

The next set of lines describes the parallelism in the workflow. We go through all the possible ruptures and extract the relevant component (*subsgt*) of the SGT mesh. Chapter 6 describes the nature of these *extract()* jobs.

```
foreach rup in rups {
  Sgt subsgt;
  subsgt = extract(sgt, rup);

  Variations vars = get_variations(site, rup);
```

For each rupture, there is an associated set of rupture variations. For this workflow, a database is queried to identify the filenames of each of the rupture variation files given the rupture ID. Each rupture variation is needed to perform seismogram synthesis and obtain peak ground accelerations. There's an average of 68.26 variations per rupture, each one is associated with a *seismogram()* and *peak\_calc()* job. The inner loop below contains 405,378 instances. This corresponds to the total number of rupture variations for the site *LGU*. The distribution of the sizes of the variations per rupture or inner loop is described in Figure 5.1

```
  Seismogram seis[];
  PeakValue peak[];
  foreach var,i in vars {
    seis[i] = seismogram(subsgt, var);
    peak[i] = peak_calc(seis[i], var);
  } // end foreach over vars
} // end foreach over rups
```

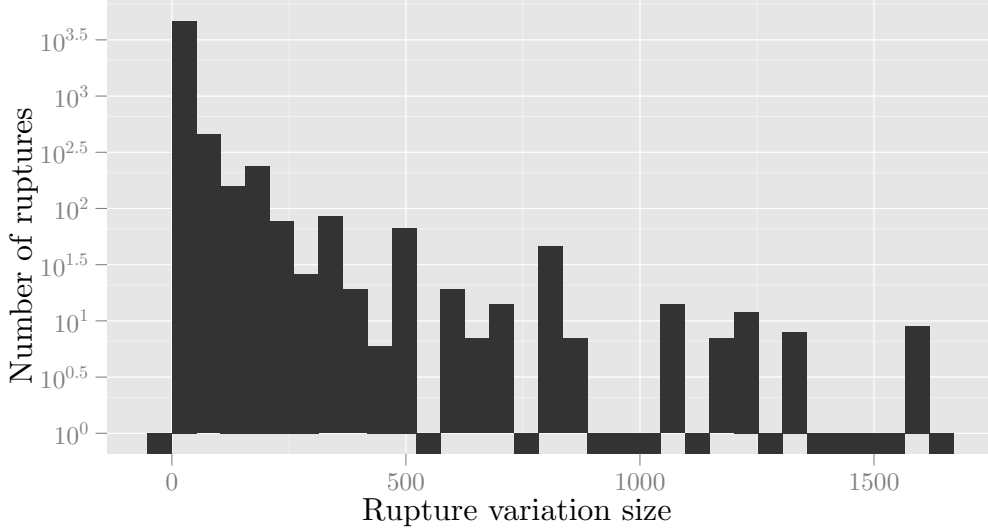


Figure 5.1: Rupture variation sizes for the site *LGU*.

## 5.4 Pilot jobs

Instead of using Swift’s default Coaster setup, we configured our site catalog to use *persistent coaster services* and *passive workers* as described in our setup in Figure 5.2. We turned off Swift’s default mechanism of bootstrapping a coaster service on a remote resource’s head node. A separate machine is dedicated to run a pool of Coaster services that persists through multiple Swift sessions. Each instance of a Coaster service on the separate machine correspond to an remote resource in TeraGrid and OSG.

For the workers, the client workflow is not involved in requesting the pilot jobs. Generating pilot jobs is independent of the number of jobs in the post-processing computation. For generating workers, we prototyped two pilot job submission mechanisms.

Our first implementation used Swift to drive submission of coaster jobs. The assumption in this setup is that the submission of workers will be regulated by Swift’s adaptive scheduler. Using the Condor-G execution provider in CoG, the generates a Condor-G submit file per worker request. The number of requests can exceed to tens of thousands of jobs. This resulted is a stressed Condor scheduler because of the volume of jobs produced, and overloaded filesystem because of a large amount of submit files.

Our second pilot submission prototype submits jobs to every remote computing resource using a fixed job submission rate. Instead of submitting one Condor submit file per worker, requests are clustered by remote resource in one file. This reduced the stress on the filesystem.

Each pilot pilot job will then register to the Coaster service corresponding to

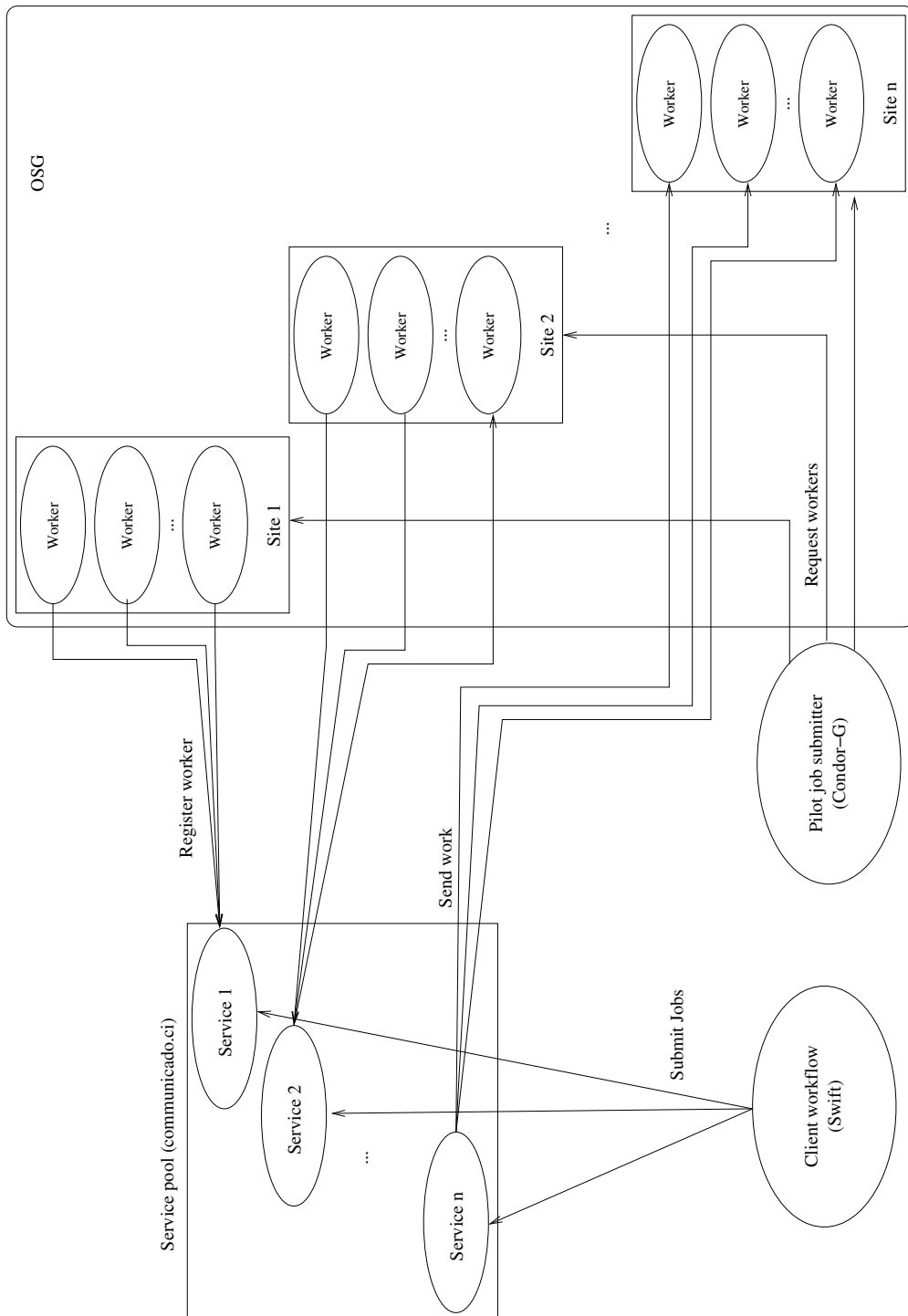


Figure 5.2: Schematic of a manual worker submission engine to a persistent coaster service.

that remote resource in the service pool. When we compute the post-processing computation, the client workflow will send a job request to a Coaster service. The Coaster service then dispatches the computation request received to one of its registered workers.

In contrast to GlideinWMS and default Swift coasters, the number of jobs requested to the compute resources are independent of the workflow. We simply submitted a factor of the number of CPUs available on the resource. Chapter 6 describes the amount of computation time obtained opportunistically in our pilot systems.

## 5.5 Globus Online

Input and output data files are managed by the Swift or Pegasus workflow systems in both implementations. The current transfer mechanisms do not support tuned file automatically tuned transfer parameters and only use the default GridFTP settings. This leads to underutilized bandwidth between the data source and destination compute resources. Handing off this component to dedicated data transfer services can aid the user in optimizing the transfer.

Globus Online is a hosted file transfer service [30]. Its main objective is to reduce the overhead to access high performance services of computing resources like those from TeraGrid and OSG. Example overheads include installation of Globus Toolkit's GridFTP client tools like `globus-url-copy` or NCSA's UberFTP. It uses automatic tuning techniques [31] to identify the ideal GridFTP transfer parameters.

We made a data transfer workload from preliminary runs of the Swift post-processing workflow. Invocations of `extract()` and `seispeak()` were extracted to identify the rupture variation files, subSGT meshes and remote resource destinations to generate the workload. To emulate the dispatching to remote resources, we replaced the program in the transformation catalog to `nop()` jobs. Essentially it is an operation that instantly returns a successful job execution.

## Chapter 6

# Performance model

We characterize a representative geographic site called *LGU*. The workflow is described in terms of the computation time and data volume requirements. Computation requirements are sampled by recording the running times of each job in the workflow. Data requirements examined by looking at the logs of how a job was invoked. The sampling of the runs were made on the PADS resource<sup>1</sup> at the Computation Institute. Accessibility to logs and other diagnostic tools makes PADS a good testbed for our experiments.

### 6.1 Compute component

We assume that the data is already in the file resource where the jobs are executed. Running times are sampled from the Swift wrapper logs to estimated the total CPU-hour consumption of the workflow. The rest of the section describes the running time distribution of each workflow segment.

The *extract()* jobs on the PADS resource. For *LGU*, there is a total of 5,939 jobs. Figure 6.1 shows the runtime distribution of the jobs. The total computation time is 121.00 CPU-hours. If we assume that we have 1,000 CPUs available on OSG, the *extract()* stage can finish in 7.26 min in a perfectly load-balanced scenario. This means at the next stage of computation, there will be 930 *seismogram()* jobs/s ready for execution.

For the moment, we have not yet ran all the 405,378 *seismogram()* jobs. But runtime recordings of 22,690 jobs were recorded in preliminary runs. We used this sample to estimate the total runtime of *seismogram()*. Using the average as the representative runtime of all 405,378 jobs, this stage consumes 17,766.93 CPU-hours.

With the finished 22,690 *seismogram()* jobs, we were able to sample 21,955 *peak\_calc()* jobs. Figure 6.3 shows the distribution of its runtime. The projected

---

<sup>1</sup>PADS is a petabyte-scale online storage system coupled with a 9 TeraFlo/s. It is mainly used for data-intensive analysis in the University. A detailed description of the system is found in [32]

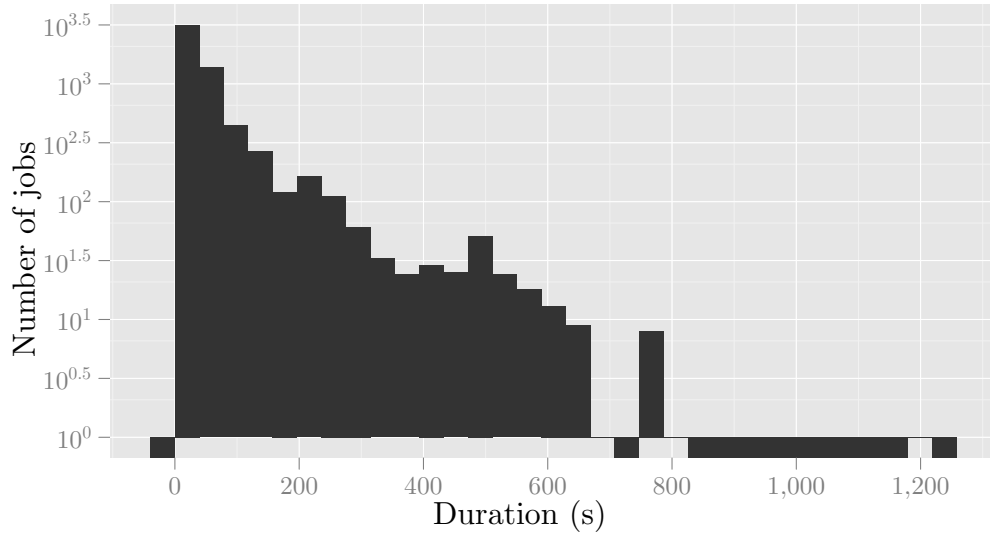


Figure 6.1: Runtime distribution of *extract()* jobs in PADS. The average runtime is 73.35 s.

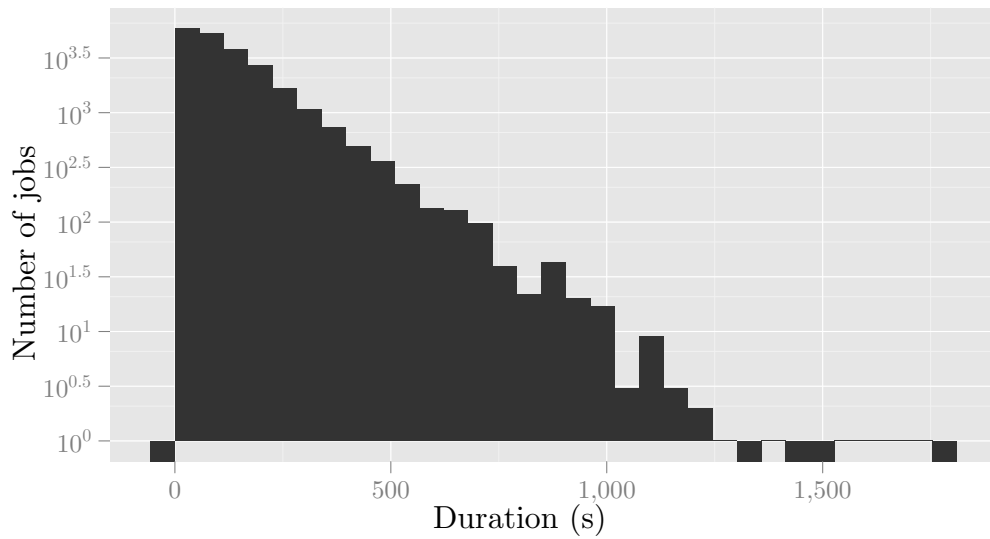


Figure 6.2: Runtime distribution of *seismogram()* from a sample of 22,690 jobs out of 405,378. The average runtime is 157.78 s.

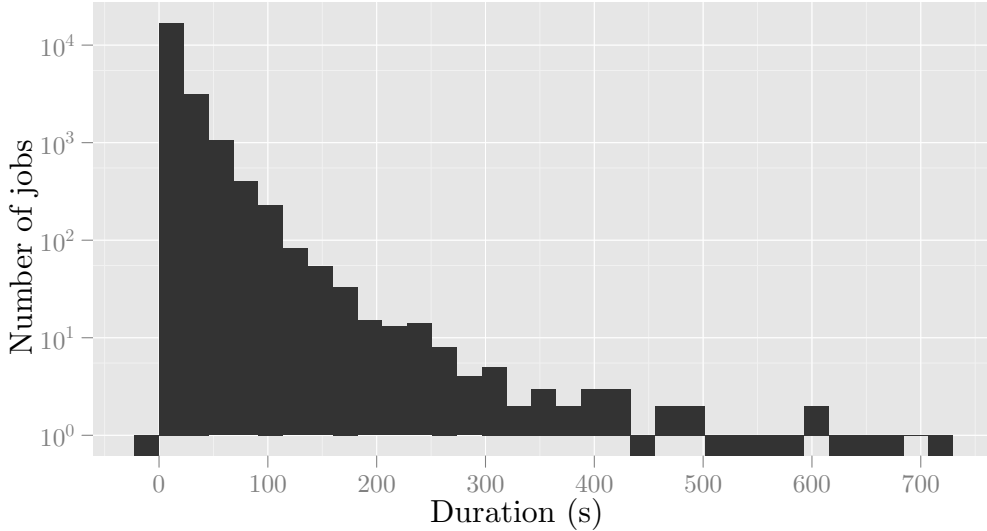


Figure 6.3: Peak ground acceleration job distribution from a sample of 21,956 jobs out of 405,378. The average runtime is 17.34 s.

runtime for 405,378 jobs is 1,952.23 CPU-hours.

Adding the runtime of all the stages required for the post-processing computation, it will take 19,840.16 CPU-hours to produce a hazard curve *LGU*. Note that the projected runtimes may diverge. Figures 6.3 and 6.2 shows a high variance in the distribution of job runtimes. So the total runtime for *LGU* may be higher or lower by  $\pm 104.60$  s (standard deviation).

In our sample runs, we observed a bottleneck in between the *seismogram()* and *peak\_calc()* jobs. By default, Swift stages out the data back to the location described in the workflow once the corresponding job finishes. In the intermediate job *seismogram()*, the seismogram data is staged out and then staged in to another compute resource. This results in too many small files being staged out. *peak\_calc()* jobs cannot run as soon as its dependency job finishes. To override this default mechanism on intermediate data, we merged the seismogram synthesis and peak acceleration calculation jobs into a job called *seispeak()*. Below is the change in our Swift implementation:

```
(seis[i], peak[i]) = seispeak(subsgt, var, site);
```

## 6.2 Opportunistic potential

To estimate the throughput that OSG can provide, generated a workload similar to the postprocessing workflow except for the data requirements. We also made

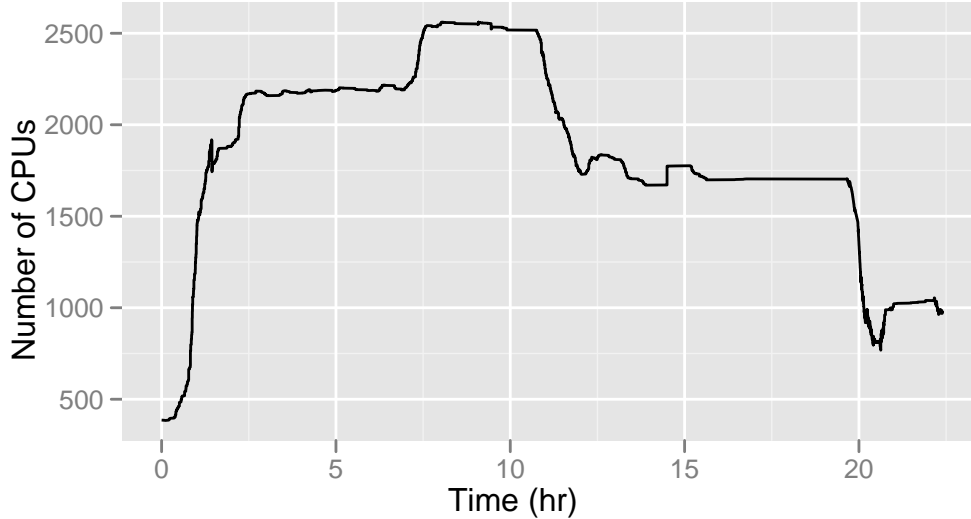


Figure 6.4: Four hour worker jobs requested on 18 OSG computing resources.

each job to have a running time of 5 min. Our results are described in terms of speedup  $S_P$ . It is defined as

$$S_P = \frac{T_1}{T_P} \quad (6.1)$$

in [33] where  $T_1$  is the sequential execution time and  $T_P$  is the parallel execution time of the our workflow. We ran a set of two experiments that lasts a day.

The first experiment used Swift’s adaptive scheduler and ran 6,000 jobs. There were Swift bugs encountered during this experiment like opening too many Condor log files. Out of 6,000 jobs, only 1,500 jobs successfully ran within  $T_P = 2,000$  s. With 5 min jobs, the serial execution time for 1,500 jobs is  $T_1 = 450,000$  s or 125 CPU-hours. Applying (6.1) the speedup for this experiment is  $S_P = 225$ .

We ran similar experiment but used RENCIT’s OSG matchmaker feature on Condor-G instead. Out of the 9000 5 min jobs sent, 7,326 ( $T_1 = 610.5$  hours) finished successfully in the  $T_P = 23.65$  hour measurement window. The speedup in for this setup is  $S_P = 25.81$ .

The rates above are not enough to finish Cybershake in its target time-to-solution time of 1–2 days. Job submission overhead is high when a workload is purely composed of short-duration jobs. Hence, pilot jobs are necessary to run the post-processing workflow to meet the required speedup. Using the two pilot job mechanisms described in Chapter 5, we requested 4 hour pilot jobs to 18 OSG resources. Within a period of a  $T_P = 24$  hours, we held at least 1,600 CPUs spread across several resources. Our run described in Figure 6.4 opened a total  $T_1 = 33,036.06$  CPU-hours for the post-processing workflow to execute.

With a speedup  $S_P = 1,361.87$  and using the total time number in Section 6.1, the post-processing computations can be completed with a time-to-solution of

$$T_P = \frac{T_1}{S_P} = \frac{19,840.16 \text{ hours}}{1,361.87} = 14.55 \text{ hours.}$$

This is within the ExTENCI target of 1–2 days.

## 6.3 Data component

In this section, we derive the data rates needed to exploit opportunistically-available OSG resources. For simplicity of our calculations, we computed target data rates to fill 2,000 CPUs with jobs. These CPUs are assumed to be spread across  $n$  resources. For input data, we assume that the SGT mesh data is housed in the PADS GridFTP server. The rupture variation files are also located in PADS.

### 6.3.1 *extract()*

The *extract()* jobs consume the 39.6 GB SGT dataset and a 864.14 kB (average) rupture variation file. This rupture variation file contains the rupture information to identify the relevant components of the strain Green’s tensors. There is a total of 5.13 GB worth of rupture variation files. Its size distribution is described in Figure 6.5. For output, each job produces an average of 163.43 MB output that is stored in 2 files. The size distribution of subSGT meshes is shown in Figure 6.6. The total size of all 5,939 subSGT meshes is 970.61 GB.

If *extract()* jobs are to be executed on the 2,000 CPU OSG resource, we need to dispatch 2,000 tasks within its average runtime of 73.35 s. This is equivalent to a job rate of

$$\frac{2,000 \text{ jobs}}{73.35 \text{ s}} = 27.27 \text{ job/s.}$$

Suppose the 39.6 GB SGT mesh is not needed in the computation. For each job, we need to send a rupture variation file with an average size of 64.14 kB in

$$\frac{1 \text{ job}}{27.27 \text{ job/s}} = 36.68 \text{ ms.}$$

Just considering staging rupture variation files, the bandwidth needed to fully utilize 2,000 CPUs for *extract()* jobs is

$$\frac{864.14 \text{ kB}}{36.68 \text{ ms}} = 23.56 \text{ MB/s.}$$

Now consider the bandwidth needed for staging in the SGT mesh. If Swift dispatches the 5,939 jobs to all the  $n$  resources, 39.6 GB will be staged  $n$  times

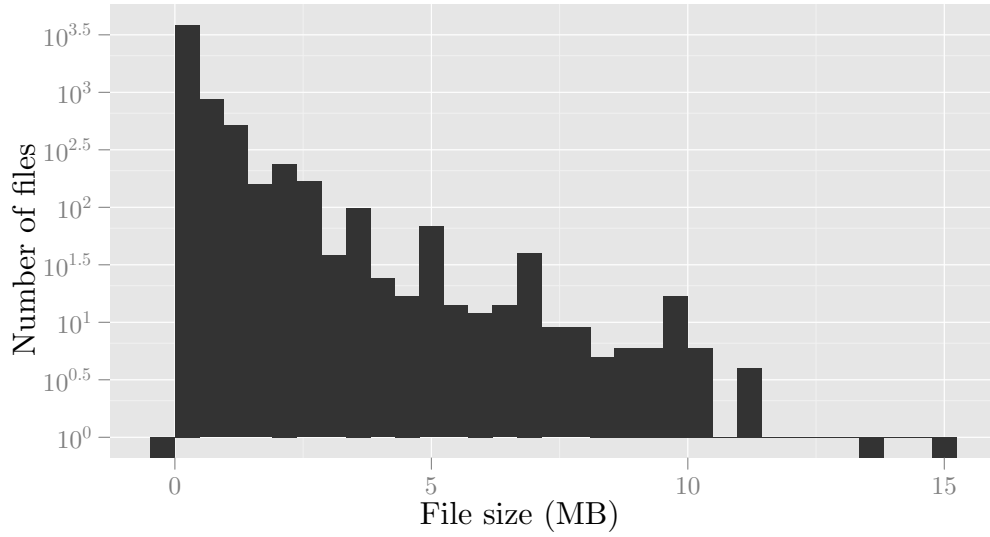


Figure 6.5: Data size distribution of rupture variation files needed by *extract()* jobs. The average size is 864.14 kB.

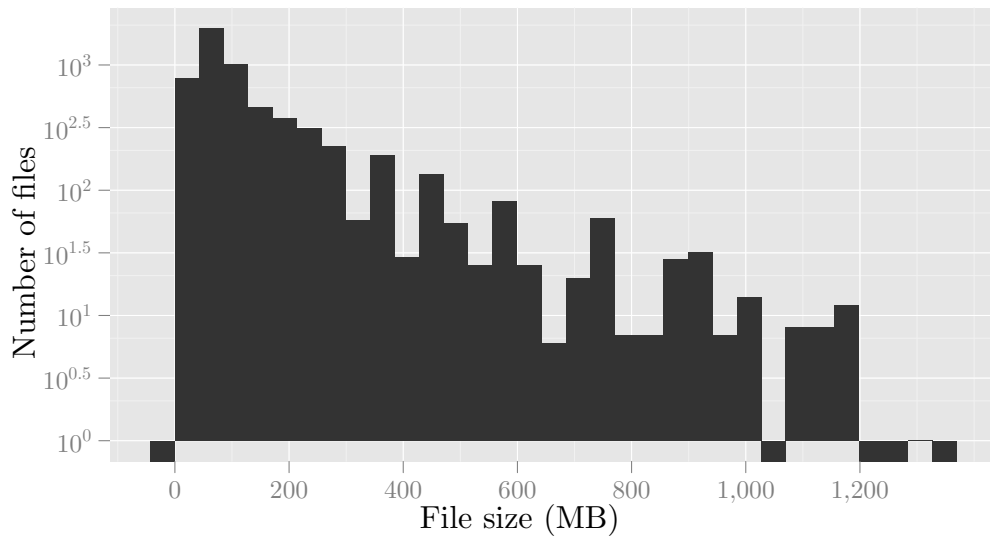


Figure 6.6: Data size distribution of subSGT meshes. The average size is 163.43 MB.

as well. If the input data is amortized across all the *extract()* jobs, each has to transfer

$$864.12 \text{ kB} + \frac{39.6 \text{ GB} \cdot n}{5,939 \text{ jobs}} = 6.67n \text{ MB/job.}$$

The fewer sites, the less bandwidth needed to fill 2,000 CPUs. For example, if all *extract()* jobs will be executed on a large OSG compute resource like *Firefly* at the University of Nebraska–Lincoln (5,000 CPUs), the data transfer required is

$$23.56 \text{ MB/s} + \frac{6.67 \text{ MB/job} \cdot 2,000 \text{ jobs}}{73.35 \text{ s}} = 205.43 \text{ MB/s.} \quad (6.2)$$

For 20 sites, the bandwidth requirement is

$$23.56 \text{ MB/s} + \frac{6.67 \cdot 20 \text{ MB/job} \cdot 2,000 \text{ jobs}}{73.35 \text{ s}} = 3.66 \text{ GB/s.}$$

Bandwidth requirements can also be calculated from a total perspective. For 2,000 CPUs, the *extract()* stage can be finished in

$$\frac{121.00 \text{ CPU-hours}}{2,000 \text{ CPUs}} = 3.63 \text{ min.}$$

For rupture variation data only, the workflow needs to stage in 5.13 GB in this time period. The bandwidth required for this scenario is

$$\frac{5.13 \text{ GB}}{3.63 \text{ min}} = 23.56 \text{ MB/s,}$$

If the SGT mesh is needed to be staged to  $n$  OSG resources, the workflow will need to stage files at the rate of

$$23.56 \text{ MB/s} + \frac{39.6n \text{ GB}}{3.63 \text{ min}} = 23.56 + 181.82n \text{ MB/s.}$$

For a single ( $n = 1$ ) OSG resource, the bandwidth needed is

$$23.56 + 181.82 \cdot 1 \text{ MB/s} = 205.38 \text{ MB/s.}$$

For  $n = 20$  resources, the requirement is 3.66 GB/s. The calculations in this approach are consistent with (6.2).

Note that calculation in the previous paragraph assumes a perfect load balance of *extract()* tasks. At the worst case, the entire computation will wait for the longest running job which is 1,181.00 s from Figure 6.1. The bandwidth requirement in this case is

$$\frac{5.13 \text{ GB} + 39.60n \text{ GB}}{1,181.00 \text{ s}} = 4.35 + 33.53n \text{ MB/s.}$$

For a single ( $n = 1$ ) OSG resource, the bandwidth needed is 37.88 MB/s. And for  $n = 20$  OSG resources, the requirement is 674.95 MB/s. The numbers in this case is an order of magnitude smaller than the perfectly load balanced case.

For both scenarios in the *extract()* stage, it is best to keep the computation on TeraGrid. Assuming an average OSG site has a 1 Gbit/s connection, a 125 MB/s bandwidth is not enough to keep 2,000 cores occupied with *extract()* jobs.

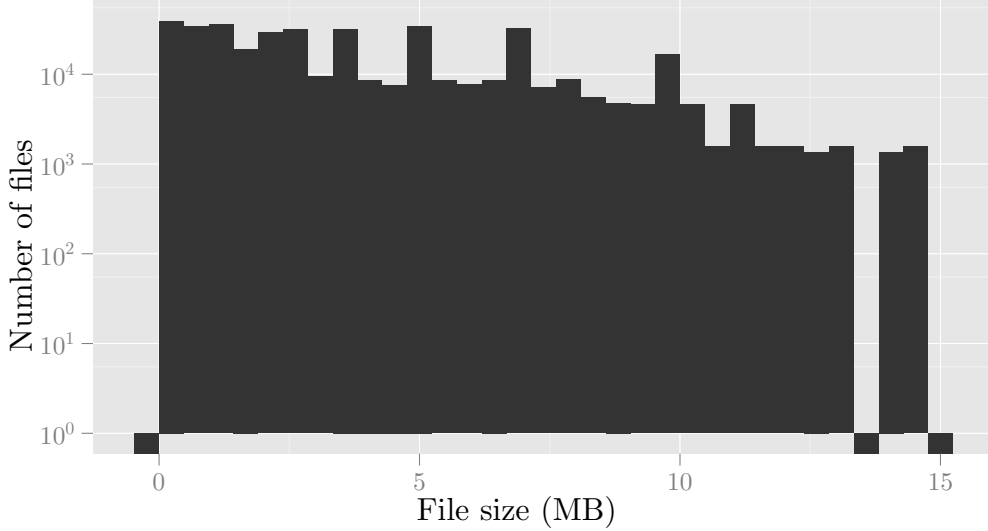


Figure 6.7: Data distribution of rupture variations needed by *seispeak()* jobs. The average size is 4.09 MB.

### 6.3.2 *seispeak()*

Each of the 405,378 *seispeak()* jobs consumes a 163.43 MB subSGT mesh and a 4.09 MB rupture variation. The total size for both data sets are 970.61 GB and 1.66 TB respectively. Figures 6.6 and 6.7 shows the data distribution of the respective input files. The data output of these jobs is a 24 kB seismogram and a 214 B spectral acceleration value. Data sizes at this stage is now uniform. The total output data expected for post-processing the geographic site *LGU* is 9.72 GB for seismograms and 86.72 MB for spectral acceleration values.

We use the the same approach in the previous subsection to get data rate targets for *seispeak()*. In this stage, 2,000 jobs are needed to be dispatched within 175.12 s. This time is obtained by adding the average runtimes of *seismogram()* and *peak\_calc()* jobs in Figures 6.2 and 6.3. The target job rate is then

$$\frac{2,000 \text{ jobs}}{175.12 \text{ s}} = 11.42 \text{ jobs/s.}$$

Assuming only rupture variation files are needed to run *seispeak()*, we need to send an average of 4.09 MB in 87.56 ms. Thus, the bandwidth needed to fully utilize 2,000 CPUs is

$$\frac{4.09 \text{ MB}}{87.56 \text{ ms}} = 46.03 \text{ MB/s.} \quad (6.3)$$

We recompute the bandwidth requirement again from a total perspective. The job *seispeak()* has a total runtime of 19,719.16 CPU-hours as computed in

Section 6.1. Using the 2,000 CPUs as the factor for speedup, the projected time it takes to compute *seispeak()* is

$$\frac{19,719.16 \text{ CPU-hours}}{2,000 \text{ CPUs}} = 9.86 \text{ hours.}$$

During this time period, we need to transfer a total of

$$4.09 \text{ MB} \cdot 405,378 = 1.66 \text{ TB}$$

for a rupture variation only scenario. The bandwidth required in this situation is

$$\frac{1.66 \text{ TB}}{9.86 \text{ hours}} = 46.77 \text{ MB/s,} \quad (6.4)$$

which is within the value computed in (6.3).

When subSGT mesh data files are included in the bandwidth calculation, the amount of data needed is

$$1.66 \text{ TB} + 163.43 \text{ MB} \cdot 5,939 = 970.61n \text{ GB} + 1.66 \text{ TB,}$$

where  $n$  is the number of OSG resources. For a single resource ( $n = 1$ ) scenario, the bandwidth requirement is

$$\frac{970.61 \text{ GB}}{9.86 \text{ hours}} + 46.77 \text{ MB/s} = 74.09 \text{ MB/s.}$$

For the worst case, all subSGT meshes are staged to all  $n$  resources. For  $n = 20$  the bandwidth requirement is 593.59 MB/s.

## 6.4 Transfer measurements

Using the process described in Section 5.5 we generated a transfer workload of 10,000 *seispeak()* jobs. The data source used is PADS and the destination is 15 OSG resources described in Table 6.1. The total data transferred is 15.49 GB. This workload was fed to the Globus Online file transfer service. For each rupture variation file  $i$ , Globus Online records the transfer completion  $t_i$  and amount of data transferred  $d(j)$ . The bandwidth  $b(t_i)$  over time is calculated as

$$b(t_i) = \frac{\sum_{j=1}^i d(j)}{t_i}. \quad (6.5)$$

Each rupture variation file successfully sent is equivalent to one job ready. To compute the job rate  $r(t_i)$ , we summed up how many files have been transferred at a given point and divide it by the elapsed time. This can be expressed as

$$r(t_i) = \frac{i}{t_i}. \quad (6.6)$$

The result of the calculations above is a plot Figure 6.8.

The peak data rate of 5.06 MB/s in Figure 6.8 can only fill

$$5.06 \text{ MB/s} \cdot \frac{1 \text{ job}}{4.09 \text{ MB}} \cdot 175.12 \text{ s} = 216 \text{ jobs}$$

for the runtime of *seispeak()*. This is 10.8 % of the target available CPUs in OSG.

For the second experiment, we included the subSGT meshes required by the 10,000 rupture variation files. The job rate is instrumented differently because *seispeak()* needs both the rupture variation and subSGT mesh. Thus the time for a job *i* to be ready in this scenario is

$$t_i = \max(v_i, x_i, y_i),$$

where  $v_i$ ,  $x_i$ , and  $y_i$  is the time when the rupture variation,  $x$  component of subSGT and  $y$  component of subSGT are transferred completely. Table 6.2 shows the new number of files including the two subSGT mesh files and the corresponding total file transfer size. Data files among multiple rupture variations are only transferred once per OSG resource. The total data staged in the workload is 424.42 GB. We added an additional statistic in Table 6.2 to show of ratio of the number of subSGT meshes to rupture variations.

For the scenario described in Figure 6.9, it took 11.94 hours to prepare the input files of 10,000 *seispeak()* jobs. But Globus Online experiences a higher peak file transfer performance because of the requirement to transfer large files. The 163.43 MB subSGT files gave enough time for the GridFTP service to identify tuned transfer parameters. If the peak bandwidth of 28.92 MB/s is sustained, the number of jobs that will finish in the period of *seispeak()*'s average running time is

$$\frac{10,000 \text{ jobs}}{14,675.66 \text{ s}} \cdot 175.12 \text{ s} = 119.33 \text{ jobs.}$$

The result is 5.97 % of the target 2,000 CPUs in OSG.

## 6.5 Data-specific plan

Given the distribution of rupture variations in Figure 5.1, we can reduce the amount of data needed to transfer to 20 OSG resources. For example, there are 3,686 out of the 5,939 ruptures that have a 2-20 variations. Intuitively, it does not make sense to transfer the 163.43 MB subSGT mesh to 20 OSG resources if there are less than 20 variations. In this section, we analyze the data requirements for different scheduling scenarios.

For our first scheduling scenario, we only execute *seispeak()* jobs on multiple OSG resources if it is part of a rupture with more than 60 variations. From Figure 5.1, there are 4,660 ruptures with less than or equal to 60 variations. This means that there is a total of  $4,660 \cdot 163.64 \text{ MB} = 761.58 \text{ GB}$  worth of subSGT data that is only needed to be staged in to OSG once. On the other

Table 6.1: Transfer workload of 10,000 rupture variation files. Data is sent from PADS to 15 OSG resources.

Resource	Number of files	Size transferred (MB)
BNL-ATLAS	1,100	1,665.44
GridUNESP_CENTRAL	664	1,053.15
LIGO_UWM_NEMO	663	961.09
Nebraska	962	1,541.25
NYSGRID_CORNELL_NYS1	533	897.70
Prairiefire	786	1,257.45
Purdue-RCAC	1,729	2,510.86
RENCI-Engagement	508	851.22
SBGrid-Harvard-East	267	474.01
SPRACE	686	1,073.44
SWT2_CPB	1,082	1,472.75
UCR-HEP	71	111.84
UMissHEP	329	522.50
UTA_SWT2	436	769.14
WQCG-Harvard-OSG	184	323.49

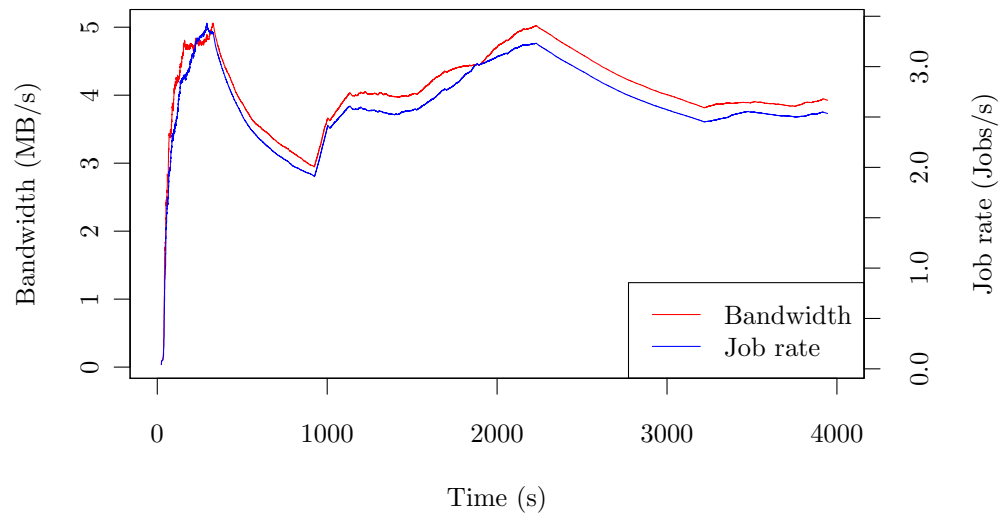


Figure 6.8: Datarate of sending rupture variation files to 15 OSG resources from PADS.

Table 6.2: Transfer workload of 10,000 rupture variation files and subSGT meshes. Data is sent from PADS to 15 OSG resource.

Resource	Number of files	Size transferred (GB)	subSGT:rupvars
BNL-ATLAS	1,370	30.71	0.25
GridUNESP_CENTRAL	986	35.39	0.48
LIGO_UWM_NEMO	993	35.99	0.50
Nebraska	1,254	33.79	0.30
NYSGRID_CORNELL_NYS1	775	28.85	0.45
Prairiefire	1,118	36.54	0.42
Purdue-RCAC	2,087	39.06	0.21
RENCI-Engagement	756	30.47	0.49
SBGrid-Harvard-East	399	17.47	0.49
SPRACE	952	31.60	0.39
SWT2_CPB	1,396	35.09	0.29
UCR-HEP	127	7.35	0.79
UMissHEP	499	22.20	0.52
UTA_SWT2	670	25.95	0.54
WQCG-Harvard-OSG	290	14.00	0.58

hand, we need to broadcast  $1,279 \cdot 163.43n \text{ MB} \cdot = 209.03n \text{ GB}$  to at most  $n$  OSG resources. In summary, only 21.54% of subSGT meshes are needed to be broadcasted. Using the bandwidth requirement for rupture variation files in (6.4), we compute the total bandwidth as

$$46.77 \text{ MB/s} + \frac{761.58 \text{ GB} + 209.03n \text{ GB}}{9.86 \text{ hours}} = 68.16 + 5.89n \text{ MB/s}$$

where  $n$  is the number of OSG resources. At the worst case where we broadcast to all  $n = 20$  resources, the total bandwidth required to drive 2,000 CPUs is 185.82 MB/s. This is only 31.33% of the original bandwidth requirement (593.59 MB/s).

In the second scenario, we add a third rule where ruptures with less than 20 variations are not executed on OSG at all. SubSGT mesh files are not needed for transfer as the corresponding *seispeak()* computations execute in TeraGrid. According to the rupture variation distribution in Figure 5.1, there are 3,686 ruptures that have less than 20 variations. The number ruptures that have variations between 20 to 60 is 974 (16.40% subSGT data, 9.25% *seispeak()* jobs). In summary 16.40% of subSGT data is staged in once compared to the previous scenario of 78.46%. Similar to the previous scenario, we still need to broadcast 21.54% of the subSGT meshes. Since some will be computed on TeraGrid, we only need to execute jobs that have more than 20 rupture variations on OSG. According to Figure 5.1 there are 371,298 rupture variations remaining.

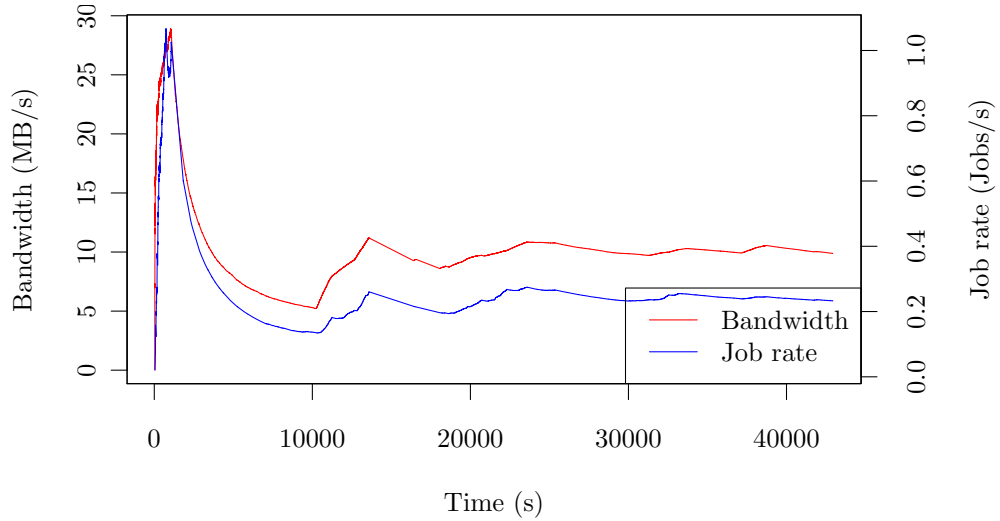


Figure 6.9: Performance of transferring subSGT meshes together with rupture variations. The total data sent over this period is 424.52 GB.

This translates to  $T_1 = 371,298 \cdot 175.12 = 18,061.58$  CPU-hours. Assuming a speedup of  $S_P = 2,000$ , the computations in OSG will complete in 9.03 hours. For rupture variation data, we need to send  $371,298 \cdot 4.09$  MB = 1.52 TB to OSG. The bandwidth requirement for this scenario is expressed as

$$\frac{1.52 \text{ TB} + (974 + 1,279n) \cdot 163.43 \text{ MB}}{9.03 \text{ hours}} = 51.65 + 6.94n \text{ MB/s.}$$

For  $n = 20$  remote OSG resources, the required bandwidth is 190.45 MB/s. This is 32.08% of the original bandwidth requirement.

## 6.6 Alternative workflow

Consider an alteration to the workflow. Instead of sending the subSGT mesh data, we send the main SGT mesh and rerun *extract()* for every *seispeak()* job. Let us call this job *postproc()*. This job will be executed 405,378 times. The new runtime for this job will be

$$175.12 \text{ s} + 73.35 \text{ s} = 248.47 \text{ s.}$$

The total runtime will increase due to extra *extract()* calculations. It will now take

$$248.47 \text{ s} \cdot 405,378 \text{ CPU} = 27,978.96 \text{ CPU-hours}$$

to complete the workflow.

For input, the new job needs the 39.6 GB SGT mesh and a 4.03 MB rupture variation file. The total data needed for this computation is expressed as

$$1.66 \text{ TB} + 39.6n \text{ GB}, \tag{6.7}$$

where  $n$  is the number of OSG resources.

To fully utilize 2,000 CPUs, the amount of data in (6.7) needs to be sent in 13.99 hours. The bandwidth required is

$$\frac{1.66 \text{ TB} + 39.6n \text{ GB}}{13.99 \text{ hours}} = 32.96 + 0.79n \text{ MB/s}.$$

This value is lower than the previous subsection because there is more time to finish staging in the 4.09 MB rupture variation file. For executing the on a single compute resource with 2,000 CPUs, the transfer rate required is only 33.76 MB/s. A single 39.6 GB SGT mesh file is smaller than 5,939 163.43 MB subSGT files so there is 54.43% less data needed to be sent to a single resource. The same argument goes for 20 OSG resources which has a bandwidth requirement of 48.65 MB/s and is only 8.19% of the original 593.59 MB/s.

## Chapter 7

# Conclusion

OSG can provide enough CPUs to Cybershake as long as the duration of the request is sufficiently long. Our initial measurements showed that requesting 4 hour pilot jobs can give the post-processing to meet the time-to-solution deadline of 1–2 days. Short job requests to OSG resources lead to low CPU acquisition yield. In addition, it stresses the job scheduler on the remote resource by sending a high number of job requests. We delegate these high volume workloads to a lightweight scheduler like the Swift Coaster service.

We deployed our own flavor of the Swift coaster system to fix minor engineering issues. This system is currently being tested by another ExTENCI group in the University to seamlessly exploit TeraGrid and OSG resources. Another fix we made was to coalesce the *seismogram()* and *peak\_calc()* jobs into a single one.

Even though there are enough CPUs in OSG to drive Cybershake’s time-to-solution requirements, data rates reported in Globus Online are not enough to drive the acquired computing resources. The bandwidth requirements of the default workflow increases proportionally with the number of OSG resources. We showed that we can regain this amount of bandwidth back by scheduling wisely. Instead of an opportunistic scheduler that is completely based on feedback from previously successful jobs, we group jobs together. This guarantees that jobs using the same dataset will be executed on the same remote resource. Recomputing the subSGT meshes can also save data transfers by one order of magnitude.

## Chapter 8

# Future work

We are evaluating was on how to schedule Cybershake on TeraGrid and OSG. Hopefully the best strategy will help complete the post-processing workflow on the geographic site *LGU*.

Explore other proposed tools in the ExTENCI project like a wide-area deployment of the Lustre filesystem. This can make the workflow more transparent to the user and avoids the engineering workarounds in the current Swift implementation.

It will also be interesting to extract the workloads and feed it into a scheduler or system simulator.

## Chapter 9

# Acknowledgements

This research was supported in part by the National Science Foundation through TeraGrid resources provided by TACC.

This research was done using resources provided by the Open Science Grid, which is supported by the National Science Foundation and the U.S. Department of Energys Office of Science

This work has been performed using the PADS resource (National Science Foundation grant OCI-0821678) at the Computation Institute a joint institute of Argonne National Laboratory and the University of Chicago.

# Bibliography

- [1] S. Callaghan, E. Deelman *et al.*, “Scaling up workflow-based applications,” *Computer and System Sciences, Journal of*, vol. 76, no. 6, p. 18, 2010. [Online]. Available: <http://dx.doi.org/10.1016/j.jcss.2009.11.005>
- [2] R. Graves, T. Jordan *et al.*, “CyberShake: A Physics-Based Seismic Hazard Model for Southern California,” *Pure and Applied Geophysics*, vol. Online Fir, pp. 1–15, May 2010. [Online]. Available: <http://www.springerlink.com/content/bv3871642p8xgnj7/>
- [3] S. Callaghan, P. Maechling *et al.*, “Reducing Time-to-Solution Using Distributed High-Throughput Mega-Workflows - Experiences from SCEC CyberShake,” in *eScience, Fourth International Conference on*, 2008, pp. 151–158. [Online]. Available: [http://ieeexplore.ieee.org/xpl/freeabs\\_all.jsp?arnumber=4736752&abstractAccess=no&userType=inst](http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4736752&abstractAccess=no&userType=inst)
- [4] L. Zhao, P. Chen, and T. Jordan, “Strain Green’s tensors, reciprocity, and their applications to seismic source and structure studies,” *Seismological Society of America, Bulletin of the*, vol. 96, no. 5, pp. 1753–1765, 2006. [Online]. Available: <http://www.bssaonline.org/cgi/content/abstract/96/5/1753>
- [5] P. Avery, R. Roskies, and D. Katz, “ExTENCI: Extending Science Through Enhanced National Cyberinfrastructure,” 2010. [Online]. Available: <http://www.nsf.gov/awardsearch/showAward.do?AwardNumber=1007115>
- [6] R. Pordes, D. Petravick *et al.*, “The open science grid,” *Physics: Conference Series, Journal of*, vol. 78, p. 012057, Jul. 2007. [Online]. Available: <http://dx.doi.org/10.1088/1742-6596/78/1/012057>
- [7] D. Reed, “Grids, the TeraGrid and beyond,” *Computer*, vol. 36, no. 1, pp. 62–68, Jan. 2003. [Online]. Available: [http://ieeexplore.ieee.org/xpl/freeabs\\_all.jsp?arnumber=1160057](http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=1160057)
- [8] E. Deelman, J. Blythe *et al.*, *Pegasus: Mapping Scientific Workflows onto the Grid*. hh: Springer Berlin / Heidelberg, 2004, vol. 3165, pp. 131–140. [Online]. Available: <http://www.springerlink.com/content/95rj5e2fgqqpkaha/>

- [9] Z. Yong, M. Hategan *et al.*, “Swift: Fast, Reliable, Loosely Coupled Parallel Computation,” in *Services, 2007 IEEE Congress on*, 2007, pp. 199–206. [Online]. Available: [http://ieeexplore.ieee.org/xpl/freeabs\\_all.jsp?arnumber=4278797&abstractAccess=no&userType=](http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4278797&abstractAccess=no&userType=)
- [10] K. G. Wilson, “Grand challenges to computational science,” *Future Generation Computer Systems*, vol. 5, no. 2-3, pp. 171–189, Sep. 1989. [Online]. Available: [http://dx.doi.org/10.1016/0167-739X\(89\)90038-1](http://dx.doi.org/10.1016/0167-739X(89)90038-1)
- [11] P. D. Lax, “Mathematics and computing,” *Journal of Statistical Physics*, vol. 43, no. 5-6, pp. 749–756, Jun. 1986. [Online]. Available: <http://www.springerlink.com/content/dr640758u803p70n/>
- [12] C. I. Council, “Cyberinfrastructure Vision for 21st Century Discovery,” p. 57, 2007. [Online]. Available: <http://www.nsf.gov/pubs/2007/nsf0728/nsf0728.pdf>
- [13] “TeraGrid - About.” [Online]. Available: <https://www.teragrid.org/web/about/>
- [14] I. Foster and C. Kesselman, *The Globus toolkit*. Morgan Kaufmann Publishers Inc., 1999, pp. 259–278. [Online]. Available: <http://www.globus.org/toolkit/>
- [15] “OSG - Members and Partners.” [Online]. Available: [http://www.opensciencegrid.org/About/Learn\\_About\\_Us/OSG\\_Organization/Members\\_and\\_Partners](http://www.opensciencegrid.org/About/Learn_About_Us/OSG_Organization/Members_and_Partners)
- [16] G. Garzoglio, T. Levshina *et al.*, *ReSS: A Resource Selection Service for the Open Science Grid*. Boston, MA: Springer US, 2009, pp. 89–98. [Online]. Available: <http://www.springerlink.com/content/m748q7212n5t7tv4/>
- [17] P. Maechling, E. Deelman *et al.*, *SCEC CyberShake Workflows Automating Probabilistic Seismic Hazard Analysis Calculations*. London: Springer London, 2007, pp. 143–163. [Online]. Available: <http://www.springerlink.com/content/x7g14n602260146q>
- [18] E. Deelman, S. Callaghan *et al.*, “Managing Large-Scale Workflow Execution from Resource Provisioning to Provenance Tracking: The CyberShake Example,” in *eScience and Grid Computing, 2006 Second IEEE International Conference on*. IEEE, 2006, pp. 14–14. [Online]. Available: [http://ieeexplore.ieee.org/xpl/freeabs\\_all.jsp?arnumber=4030987&abstractAccess=no&userType=inst](http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=4030987&abstractAccess=no&userType=inst)
- [19] G. Mehta, “CyberShake on OSG,” 2010. [Online]. Available: <https://confluence.pegasus.isi.edu/display/pegasus/CyberShake+on+OSG>
- [20] P. Couvares, T. Kosar *et al.*, *Workflow Management in Condor*. London: Springer London, 2007, pp. 357–375. [Online]. Available: <http://www.springerlink.com/content/r6un6312103m47t5/>

- [21] I. Sfiligoi, D. Bradley *et al.*, “The Pilot Way to Grid Resources Using glideinWMS,” in *Computer Science and Information Engineering, 2009 WRI World Congress on*, 2009, pp. 428–432. [Online]. Available: [http://ieeexplore.ieee.org/xpl/freeabs\\_all.jsp?arnumber=5171374&abstractAccess=no&userType=](http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5171374&abstractAccess=no&userType=)
- [22] D. Bradley, I. Sfiligoi *et al.*, “Scalability and interoperability within glideinWMS,” *Journal of Physics: Conference Series*, vol. 219, no. 6, p. 062036, Apr. 2010. [Online]. Available: <http://stacks.iop.org/1742-6596/219/i=6/a=062036>
- [23] E. Walker, J. Gardner *et al.*, “Creating Personal Adaptive Clusters for Managing Scientific Jobs in a Distributed Computing Environment,” in *Challenges of Large Applications in Distributed Environments CLADE 06, IEEE International Workshop on*, 2006.
- [24] I. Raicu, Y. Zhao *et al.*, “Falkon: a Fast and Light-weight task EXECUTION framework,” in *Supercomputing, SC 07 Proceedings of the 2007 ACM/IEEE conference on*. ACM, 2007, pp. 1–12. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1362680>
- [25] G. Banga, P. Druschel, and J. C. Mogul, “Resource Containers: A New Facility for Resource Management in Server Systems,” in *Operating Systems Design and Implementation*. USENIX Association, 1999, pp. 45–58. [Online]. Available: <http://portal.acm.org/citation.cfm?id=296810>
- [26] M. Hategan, J. Wozniak, and M. Wilde, “Coasting Through the Clouds,” 2011.
- [27] L. Moreau, Y. Zhao *et al.*, “XDTM: The XML Data Type and Mapping for Specifying Datasets,” *Advances in Grid Computing EGC 2005*, vol. 3470, pp. 495–505, 2005. [Online]. Available: [http://dx.doi.org/10.1007/11508380\\_51](http://dx.doi.org/10.1007/11508380_51)
- [28] G. Laszewski and M. Hategan, “Workflow Concepts of the Java CoG Kit,” *Grid Computing, Journal of*, vol. 3, no. 3-4, pp. 239–258, Jan. 2006. [Online]. Available: <http://www.springerlink.com/content/100337801357p38n>
- [29] M. Daydé, J. Palma *et al.*, *An Opportunistic Algorithm for Scheduling Workflows on Grids*. Springer Berlin / Heidelberg, 2007, vol. 4395, pp. 1–12. [Online]. Available: <http://www.springerlink.com/content/b3n1486r21g46633/>
- [30] I. Foster, “What the cloud really means for science,” in *Cloud Computing Technology and Science, 2nd*, Indianapolis, 2010. [Online]. Available: <http://www.slideshare.net/ianfoster/cloud-com-foster-december-2010><http://salsahpc.indiana.edu/CloudCom2010><http://www.computer.org/portal/web/csmediacenter/cloudcom2010>

- [31] W. Liu, B. Tieman *et al.*, “A data transfer framework for large-scale science experiments,” in *High Performance Distributed Computing - HPDC '10, Proceedings of the 19th ACM International Symposium on*. New York, New York, USA: ACM Press, Jun. 2010, p. 717. [Online]. Available: <http://portal.acm.org/citation.cfm?id=1851476.1851582>
- [32] “PADS Petascale Active Data Store.” [Online]. Available: <http://pads.ci.uchicago.edu/about/>
- [33] R. Scott, T. Clark, and B. Bagheri, *Parallel Performance*. Princeton University Press, 2005, pp. 37–70. [Online]. Available: <http://espace.library.uq.edu.au/view/UQ:190899>