

# International Journal of High Performance Computing Applications

<http://hpc.sagepub.com/>

---

## **Metrics for heterogeneous scientific workflows: A case study of an earthquake science application**

Scott Callaghan, Philip Maechling, Patrick Small, Kevin Milner, Gideon Juve, Thomas H Jordan, Ewa Deelman, Gaurang Mehta, Karan Vahi, Dan Gunter, Keith Beattie and Christopher Brooks

*International Journal of High Performance Computing Applications* published online 29 June 2011

DOI: 10.1177/1094342011414743

The online version of this article can be found at:

<http://hpc.sagepub.com/content/early/2011/06/27/1094342011414743>

---

Published by:



<http://www.sagepublications.com>

**Additional services and information for *International Journal of High Performance Computing Applications* can be found at:**

**Email Alerts:** <http://hpc.sagepub.com/cgi/alerts>

**Subscriptions:** <http://hpc.sagepub.com/subscriptions>

**Reprints:** <http://www.sagepub.com/journalsReprints.nav>

**Permissions:** <http://www.sagepub.com/journalsPermissions.nav>



# Metrics for heterogeneous scientific workflows: A case study of an earthquake science application

The International Journal of High Performance Computing Applications 1–12

© The Author(s) 2011

Reprints and permissions:

sagepub.co.uk/journalsPermission.nav

DOI: 10.1177/1094342011414743

hpc.sagepub.com



Scott Callaghan<sup>1</sup>, Philip Maechling<sup>1</sup>, Patrick Small<sup>1</sup>, Kevin Milner<sup>1</sup>, Gideon Juve<sup>1</sup>, Thomas H Jordan<sup>1</sup>, Ewa Deelman<sup>2</sup>, Gaurang Mehta<sup>2</sup>, Karan Vahi<sup>2</sup>, Dan Gunter<sup>3</sup>, Keith Beattie<sup>3</sup>, and Christopher Brooks<sup>4</sup>

## Abstract

Scientific workflows are a common computational model for performing scientific simulations. They may include many jobs, many scientific codes, and many file dependencies. Since scientific workflow applications may include both high-performance computing (HPC) and high-throughput computing (HTC) jobs, meaningful performance metrics are difficult to define, as neither traditional HPC metrics nor HTC metrics fully capture the extent of the application. We describe and propose the use of alternative metrics to accurately capture the scale of scientific workflows and quantify their efficiency. In this paper, we present several specific practical scientific workflow performance metrics and discuss these metrics in the context of a large-scale scientific workflow application, the Southern California Earthquake Center CyberShake 1.0 Map calculation. Our metrics reflect both computational performance, such as floating-point operations and file access, and workflow performance, such as job and task scheduling and execution. We break down performance into three levels of granularity: the task, the workflow, and the application levels, presenting a complete view of application performance. We show how our proposed metrics can be used to compare multiple invocations of the same application, as well as executions of heterogeneous applications, quantifying the amount of work performed and the efficiency of the work. Finally, we analyze CyberShake using our proposed metrics to determine potential application optimizations.

## Keywords

large-scale simulations, performance metrics, scientific workflows

## 1. Introduction

Scientific workflows compose and execute a series of computational or data manipulation tasks in a scientific application (Taylor et al., 2006). By automating distributed, multi-stage, high-performance computing (HPC) calculations, scientific workflows can improve the reliability and increase the scale of research applications. Scientific workflows may be heterogeneous, including many jobs, many scientific codes, many data files, and their dependencies. Scientific workflows may include both HPC elements, such as large-scale parallel jobs, and numerous short-running jobs typical of high-throughput computing (HTC). Jobs may communicate via message passing or be coupled via file system dependencies.

Scientific workflow performance can be difficult to characterize with standard computational metrics, because workflows may include both HPC and HTC elements, and neither traditional HPC metrics nor HTC metrics fully

capture the performance of the application. Understanding application performance includes understanding performance at both the computation and workflow levels. In this paper, we describe a number of alternative workflow performance metrics and propose their use to accurately capture the scale of scientific workflow simulations and quantify their efficiency. Such metrics can be used to reduce time to solution for research calculations, and support comparisons between scientific workflows.

<sup>1</sup>University of Southern California, USA

<sup>2</sup>USC Information Sciences Institute, USA

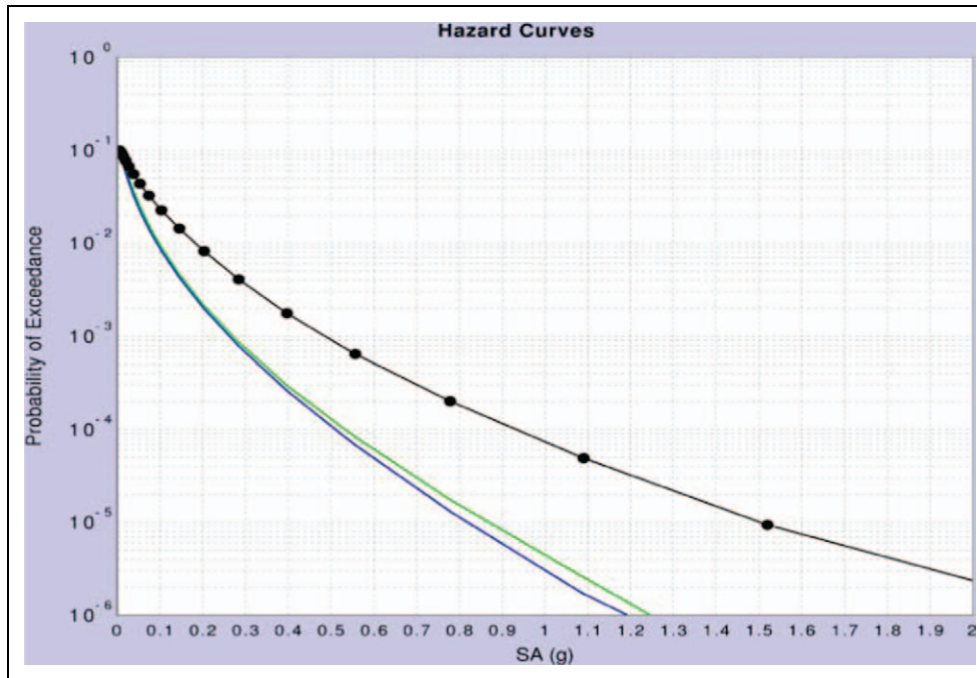
<sup>3</sup>Lawrence Berkeley National Laboratory, USA

<sup>4</sup>University of San Francisco, USA

## Corresponding author:

Scott Callaghan, University of Southern California, 3651 Trousdale Parkway, Los Angeles, CA 90089, USA

Email: scottcal@usc.edu



**Figure 1.** Three hazard curves for the Whittier Narrows site near Los Angeles. The black line is the hazard curve calculated with the physics-based CyberShake simulation. The blue and green lines are hazard curves calculated using two common attenuation relationships (color online only).

We illustrate the use of our proposed metrics by using them to describe a specific heterogeneous scientific workflow application, the Southern California Earthquake Center (SCEC; <http://www.scec.org>) CyberShake probabilistic seismic hazard computational system. Section 2 describes the science and computational requirements of CyberShake. In Section 3 we present our proposed metrics for scientific workflows. Section 4 analyzes these metrics as they apply to the CyberShake project. Section 5 gives an overview of related work. Finally, we conclude in Section 6 and invite others to use the proposed metrics for comparison of scientific workflow applications.

## 2. CyberShake

### 2.1. Science description

Seismologists quantify the seismic hazard for a location using probabilistic seismic hazard analysis (PSHA). PSHA provides a technique for estimating the probability that earthquake ground motions at a location of interest will exceed some intensity measure (IM), such as peak ground velocity or peak ground acceleration, over a given time period. These kinds of estimates are useful for civic planners, building engineers, and insurance agencies and, as the basis for building codes, influence billions of dollars of construction each year.

PSHA estimates are delivered by hazard curves that relate ground motion to probability of exceedance for a site of interest (Figure 1). A set of hazard curves for a region can be interpolated to produce a hazard map for an area,

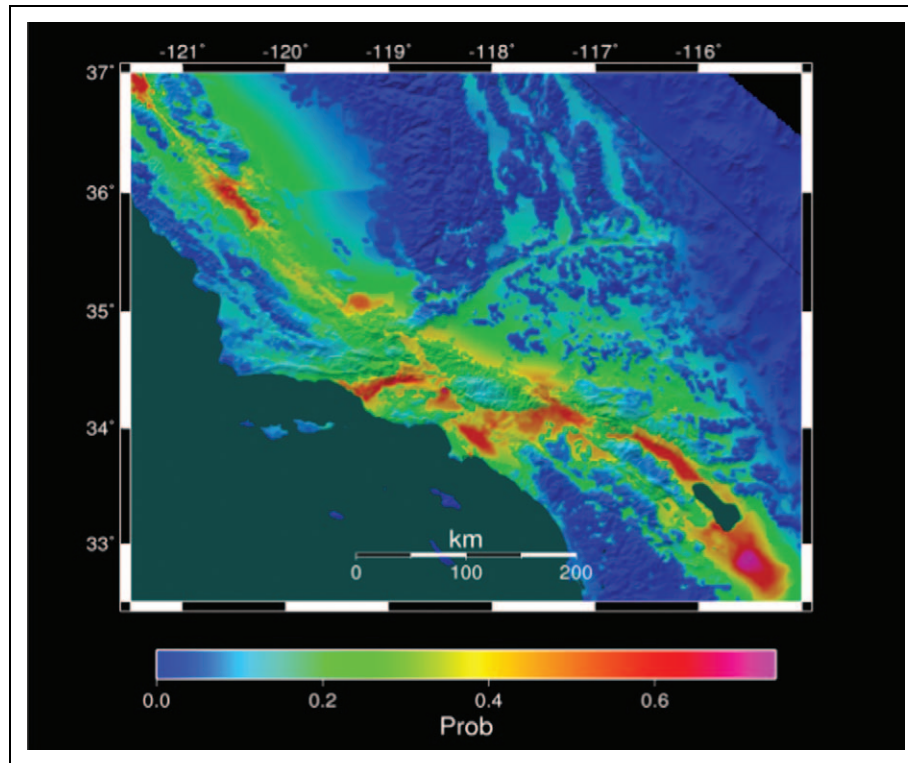
holding either probability or ground motion constant and plotting the other as a function of geographic location (Figure 2).

To produce a hazard curve for a site of interest, CyberShake must consider a set of possible large earthquakes that could influence the site. For a typical site, CyberShake considers about 7000 ruptures – that is, fault/magnitude combinations. We vary the hypocenters and slip distribution of the ruptures to produce about 415,000 different rupture variations, each of which represents a potential earthquake.

Once the ruptures and rupture variations are defined, CyberShake uses an anelastic wave propagation simulation to generate strain Green's tensors (SGTs), describing the stresses and strains in a volume around the site of interest. Seismic reciprocity is then used to post-process the SGTs and obtain synthetic seismograms, one for each rupture variation (Zhao et al., 2006). The synthetic seismograms are distilled into IM values and these are then aggregated into a single hazard curve. CyberShake hazard curves from hundreds of sites are combined into a physics-based seismic hazard map for the region, improving our understanding of seismic hazard in Southern California (Graves et al., 2008).

### 2.2. Computational description

To produce a hazard curve for a single site, CyberShake requires significant computational resources (Callaghan et al., 2008). An outline of the computational and data requirements is given in Table 1.



**Figure 2.** An example hazard map for the Southern California area generated using attenuation relationships. Colors show the probability that ground motions will exceed  $0.1g$  (acceleration due to gravity) from 2007 to 2057 (color online only).

**Table 1.** Data and central processing unit (CPU) requirements for the CyberShake components, per site of interest.

Component	Code type	Data	CPU hours
Mesh generation	Sequential	15 GB	150
SGT simulation	MPI	40 GB	12,000
SGT extraction	MPI	690 GB	250
Seismogram synthesis	Sequential	10 GB	1600
PSA calculation	Sequential	90 MB	100
Totals		755 GB	14,100

SGT: strain Green's Tensor, MPI: message-passing interface, PSA: Peak Spectral Acceleration.

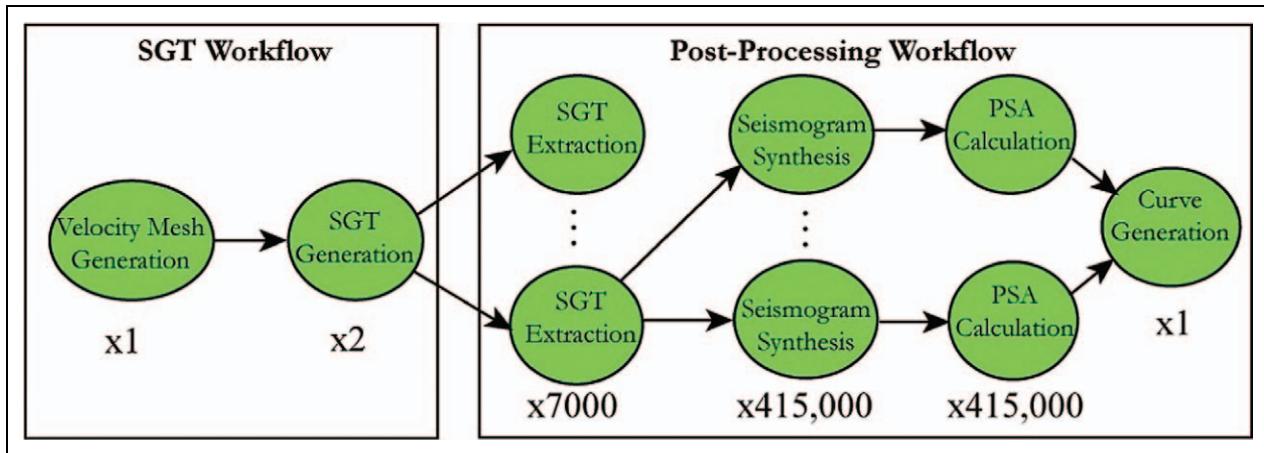
To compute the SGTs, a mesh of about 1.5 billion points is constructed and populated with seismic velocity information for the region of interest. The mesh is then used in a pair of wave propagation simulations, each for 20,000 timesteps, that establish a quantitative relationship between motion along faults in the region and ground motion at the site of interest.

After SGT generation is complete, post-processing is performed. For each rupture variation, SGTs corresponding to the location of the rupture surface are extracted from the volume. Seismic reciprocity is used to generate synthetic seismograms, which represent the ground motions that the rupture variation would produce at the site of interest. The seismograms are processed to obtain the IM of interest – peak spectral acceleration – at various periods. The execution of these post-processing steps takes only a few seconds

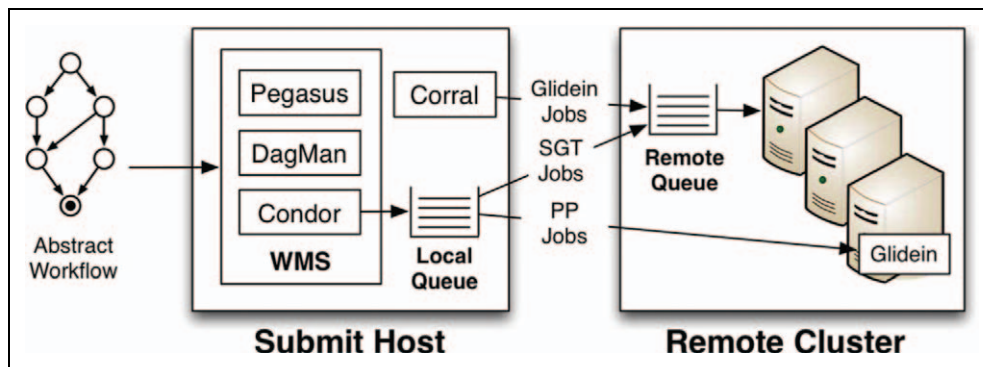
to minutes each, but SGT extraction must be performed for each of 7000 ruptures and seismogram synthesis and peak spectral acceleration processing must be executed for each of 415,000 rupture variations. Therefore, calculating a hazard curve for a single site requires about 840,000 task executions, 14,000 central processing unit (CPU) hours, and generates about 750 GB of data.

The extensive computational requirements and the large number of independent post-processing jobs necessitate a high degree of automation, to submit jobs, manage data, and provide error recovery when jobs fail. The mesh creation and SGT generation stages consist of a few large message-passing interface (MPI) jobs that run on a cluster using spatial decomposition across the processors. In contrast, the post-processing consists of hundreds of thousands of loosely coupled, semi-independent jobs. In order to manage these different stages efficiently, CyberShake is divided into two workflows, one for the SGT generation and one for post-processing. Schematics of the two workflows are presented in Figure 3.

To execute the workflows on grid resources, CyberShake uses a software stack including the Pegasus Workflow Management System (Deelman et al., 2005; <http://pegasus.isi.edu>) and Condor (<http://www.cs.wisc.edu/condor>) (Figure 4). This software enables us to create a generic, high-level workflow description, plan it for execution on a specific computational resource, and submit and monitor the jobs remotely. More details on the CyberShake architecture can be found in Callaghan et al. (2010).



**Figure 3.** Schematic for CyberShake workflows. The numbers at the bottom of the workflow stages signify the number of different tasks at each step. SGT: strain Green's Tensors, PSA: Peak Spectral Acceleration.



**Figure 4.** Execution architecture. SGT: strain Green's Tensors, WMS: workflow management system.

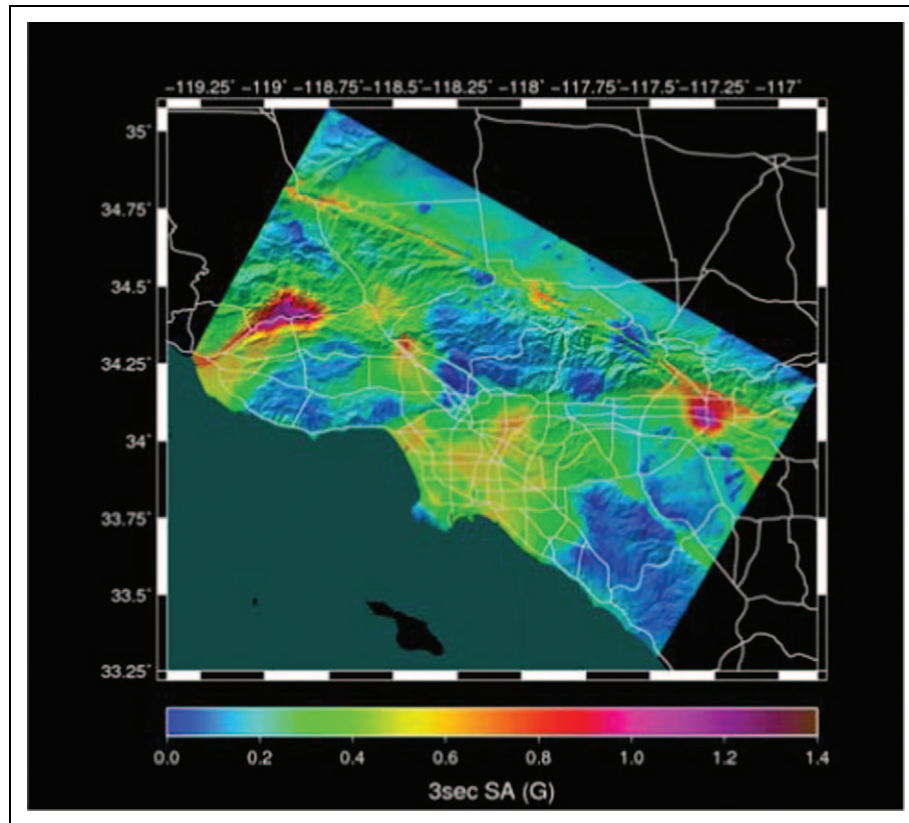
Jobs in the SGT workflow are submitted directly to the batch scheduler on the remote system using grid protocols. However, using a similar approach for the post-processing workflow resulted in poor performance due to a number of factors. Firstly, grid submission overheads and queue waiting times greatly exceeded the runtimes of the jobs themselves, resulting in low throughput. In addition, we found that many computational resources have scheduling policies that limit the number of jobs that can be queued simultaneously, resulting in poor parallelism for serial jobs. Finally, we found that submitting a large number of short-duration jobs may overwhelm the remote scheduler, resulting in job failures and other errors.

In order to achieve good performance for the post-processing workflow, CyberShake uses two optimization techniques: task clustering and resource provisioning.

When an abstract workflow is planned, normally the mapping of workflow tasks to grid jobs is one to one. Using task clustering, Pegasus is able to group multiple tasks into a single grid job. This improves the performance of the post-processing workflow by reducing the number of jobs to be executed and increasing the average runtime of the jobs, which reduces the relative scheduling overhead.

Resource provisioning allows CyberShake to allocate resources ahead of workflow execution and assume responsibility for matching jobs with resources. Provisioning for CyberShake is enabled using Condor glideins (<http://www.cs.wisc.edu/condor/glidein>) managed by Corral (<http://pegasus.isi.edu/corral/>). To provision resources, a glidein request is submitted to the remote scheduler. Instead of running an application job, the glidein reserves the resource and starts a condor daemon that can fetch application jobs directly from the workflow management system. Using this approach bypasses the remote scheduler and reduces scheduling overhead. It also enables one glidein job to provision multiple resources for a long period, which can be used to execute many workflow jobs. Finally, giving the workflow system responsibility for resource management enables CyberShake to use custom scheduling policies that result in better performance. Corral automates the complex steps required to create glidein jobs and automatically resubmits glideins as they expire, allowing CyberShake to maintain a fixed-size resource pool over a long period of time.

The combination of task clustering enabled by Pegasus and resource provisioning enabled by Corral increases throughput scalability and parallelism, and



**Figure 5.** CyberShake hazard map interpolated on attenuation base map, showing spectral acceleration levels that have a 2% probability of exceedance in 50 years. The units are spectral acceleration at a 3 second period, measured in terms of  $g$  (acceleration due to gravity) (color online only).

allows CyberShake to run on computational resources that would otherwise be difficult or impossible to use.

### 2.3. CyberShake Map 1.0 calculation

In spring 2009 we performed a CyberShake production run, the CyberShake 1.0 Map calculation. Executed on the Ranger cluster at the Texas Advanced Computing Center (<http://www.tacc.utexas.edu>) and on the University of Southern California's (USC's) High Performance Computing and Communication's (HPCC's) cluster (<http://www.usc.edu/hpcc>), this calculation used over 5 million CPU hours to execute 189 million tasks over 54 days to generate hazard curves for 223 sites around Southern California and create the world's first physics-based PSHA maps for Southern California. Figure 5 shows the hazard map generated by CyberShake, which depicts the level of shaking expected with 2% probability in 50 years. Warm colors represent areas of higher shaking. Figure 6 illustrates the difference between CyberShake and traditional attenuation hazard maps. Each dot represents a location for which CyberShake generated hazard curves. Red indicates regions where CyberShake calculated higher hazard, blue lower. Interestingly, many of the areas where CyberShake found higher hazard coincide with areas of high population density in the Los Angeles basin, underscoring the importance of accurately determining seismic hazard.

One of the challenges of describing this large production run is to present metrics that meaningfully capture the scope of the simulation and permit straightforward comparison to other heterogeneous scientific workflows. Next we will discuss our metrics.

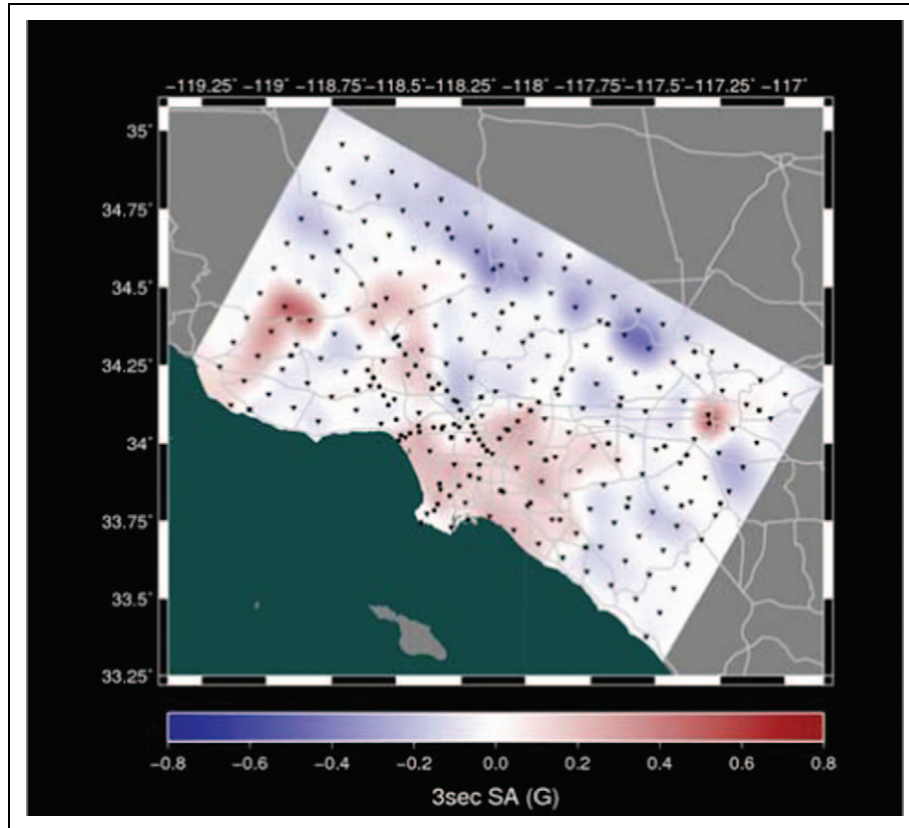
### 3. Scientific workflow performance metrics

The goal of performance metrics is to provide a technique for comparing application executions and identifying potential optimizations. This includes multiple invocations of the same application, as well as executions of multiple applications. Therefore, proposed metrics must provide an application-independent way of comparing performance for heterogeneous applications. In addition, a relatively small set of metrics is desired to simplify comparisons.

We use performance metrics to answer two fundamental questions:

- How much useful work was performed?
- How efficiently was the work performed?

For large-scale workflows, there are two types of 'work' to consider. One is computational work, relating to



**Figure 6.** This map shows the differences between the CyberShake and attenuation hazard maps. Red represents higher CyberShake hazard, blue represents lower (color online only).

floating-point operations (FLOPs), file access, and parallelism. The other, more difficult to characterize, is workflow work, describing the jobs and tasks that need to be coordinated, scheduled, and executed. Both are important for understanding the size and performance of a scientific workflow.

We have divided our proposed metrics into three categories based on scale: *task level*, *workflow level*, and *application level*. Breaking down the measures this way makes it easy to understand what factors are contributing to each aspect of performance.

### 3.1. Task-level metrics

Task-level metrics report performance for individual executions. At this level, only computational metrics are meaningful. The magnitude of work can be characterized by traditional metrics and throughput measures, such as:

1. *FLOPs, FLOPs/sec;*
2. *File input/output (I/O), file I/O/sec;*
3. *Task runtime.*

These metrics are commonly used and are well understood to quantify the computational requirements of a task. They can be captured at runtime by the use of the

Performance Application Programming Interface (PAPI; Dongarra et al., 2001) or other low-level performance tools.

### 3.2. Workflow-level metrics

Workflow-level metrics describe aggregate performance for an entire workflow. At this level, we can determine the overall computational work and investigate the performance of the workflow tools.

To quantify the computational work, we can combine the job-level metrics to get the following aggregate measures:

4. *Workflow FLOP count, FLOP/sec;*
5. *Workflow file I/O, file I/O/sec;*
6. *Workflow compute time, defined as the sum over all the tasks of the product of task runtime and the number of CPUs used by the task;*
7. *Workflow makespan, defined as the amount of elapsed walltime from when the first job in the workflow is submitted until the last job completes.*

These metrics describe the demands of a scientific workflow on the execution resource.

Describing the workflow load of a scientific workflow application on the workflow tools is less clear. Since a multi-layered software stack is required to support

workflow execution, multiple components contribute to performance. Although these components can be analyzed independently in a testbed with dummy jobs, it can be difficult to accurately predict how the various middleware layers will interact with real jobs in a production environment. We propose the following metrics to quantify how these elements contribute to the overall performance of the workflow, using quantities measurable at execution time.

To determine how much useful workflow work was performed, we suggest the following.

8. *Number of tasks and tasks executed/sec.* We define a task as an execution unit on a computational resource; that is, a single invocation of a scientific code. Analogous to FLOPs for computational work, the number of tasks required for a simulation is dictated by the science requirements, and provides a way to determine how much load the workflow tools must handle.

To measure workflow efficiency, a metric must relate actual performance to possible peak performance. It is important to recognize that the work unit for a workflow scheduler may be different than the tasks discussed previously (see discussion in Section 2.2). To reflect this distinction, we define a job as an execution unit for which a scheduler must manage dependencies and schedule to available resources. Jobs may consist of one or more tasks.

9. *Delay per job.* Delay per job is the average, over all workflow jobs, of the elapsed time between when the workflow tools have determined that a job was first available to run (all its dependencies have completed successfully) and the time when the job actually started on the remote resource.

Ideally, the delay per job would be 0, indicating that each job started running as soon as all its dependencies had completed. In reality, the delay per job is influenced by the availability of computational resources and the overhead of workflow tools in scheduling jobs to those resources. If no resources are available, the delay will accumulate. It also reflects the efficiency of matching resources to jobs and dispatching jobs to execute. Delay per job is meaningful for both advanced resource provisioning and jobs submitted to a remote batch scheduler, and reflects overhead and delay in both cases. Depending on the specific middleware involved, delay per job could be broken down into more detailed phases (Nerieri et al., 2006; Prodan and Fahringer, 2008).

We suggest an analogous measure to HPC parallel efficiency to quantify the parallelism of a workflow. Since a scientific workflow may contain both serial jobs, which can be independently run in parallel as well as MPI parallel jobs, a new metric is needed,

10. *Workflow parallel speedup*, defined as the ratio of the workflow compute time to the makespan of the

workflow. This is a logical extension of the SpeedUp-1 (SU-1) metric proposed for serial jobs by Ostermann et al. (2008) that can be applied to both parallel and serial jobs. It reflects both parallelism obtained by running multicore jobs, as well as parallelism via simultaneous execution of multiple serial jobs.

### 3.3. Application-level metrics

We define a workflow application as the set of calculations needed to obtain the desired research result. Not all scientific workflow applications consist of a single workflow. Some may include multiple workflows, or multiple executions of a single workflow. For example, a parameter sweep may consist of hundreds of similar workflows. We feel it is valuable to capture the full extent of a multi-workflow application; thus we introduce application-level metrics, which aggregate results for individual scientific workflows that make up the entire application. Computationally, this includes:

11. *Total FLOP count, FLOP/sec;*
12. *Total file I/O, file I/O/sec;*
13. *Application compute time, defined as the sum of the compute times of the constituent workflows.*

Some of these metrics may seem repetitive and similar to values at the workflow level. However, since an application may consist of multiple concurrent workflows, examining metrics at the application level may reveal scalability bottlenecks not otherwise captured.

Application-level workflow metrics include the aggregate measures:

14. *Total number of tasks, tasks executed/sec;*
15. *Total number of jobs, jobs executed/sec.*

Along with the number of completed tasks, we look at the following.

16. *Total number of failed jobs.* These are jobs that failed for any reason and needed to be retried or aborted. Looking at failed jobs gives a sense of the reliability of both the software and hardware.

The workflow-level metrics can also be averaged over all workflows in the application. Much like workflow parallel speedup, we can define a measure of application-level parallelism as follows:

17. *Application parallel speedup*, as the ratio of application compute time to the makespan of the application.

Since multiple workflows may be run at once, application parallel speedup reflects the parallelism of multiple workflows, as well as the parallelism within workflows.

**Table 2.** CyberShake task-level metrics.

Task type	Task runtime		I/O read		I/O write		Peak memory		CPU utilization	
	Mean (s)	Std. Dev.	Mean (MB)	Std. Dev.	Mean (MB)	Std. Dev.	Mean (MB)	Std. Dev.	Mean (%)	Std. Dev.
ExtractSGT	110.58	141.20	175.40	177.70	155.86	176.48	20.64	0.64	65.82	0.28
ZipPeakSA	195.80	237.99	1.07	0.07	2.26	0.15	6.16	0.16	2.89	0.01
PVCOkaya	0.55	2.48	0.02	0.00	0.00	0.00	3.11	0.01	16.89	0.04
ZipSeis	265.73	275.04	118.95	7.88	101.05	14.36	6.25	0.16	6.83	0.01
SeisSynth	79.47	70.86	547.16	324.92	0.02	0.00	817.59	483.51	92.01	0.08

I/O: input/output, CPU: central processing unit

At the application level, we can analyze workflow efficiency. Workflow efficiency is how effectively available, provisioned resources were used. This is especially applicable for a scientific application consisting of a large number of small jobs, since obtaining resources may be a separate process from using those resources. As a result, it is possible that not all resources may be used at all times. This may be a result of the structure of the workflows; the workflow graphs may have bottlenecks where not enough parallelism exists to use all available resources. In addition, task management systems, such as Condor, schedule to resources at a fixed rate, which may be insufficient to keep all resources in use. We quantify this as follows.

18. *Resource utilization* is defined as the ratio of the application compute time to the CPU time of provisioned resources. This does not include queue time waiting for resources.

Ideally, 100% resource utilization would indicate that all provisioned resources were always used. More realistically, resource utilization will be affected by the inherent parallelism in the workflow structure and the overhead in job scheduling. High resource utilization is important both for reducing end-to-end time by maximizing parallelism and for minimizing idle cycles.

Obviously resource utilization is dependent on resource availability and whether dynamic or static resource provisioning is used. If few resources are available, resource utilization could be high but application parallel speedup low, leading to long wallclock time. As with all metrics, taking any one by itself can be misleading; they should be considered together.

When defining metrics for workflows, one issue to consider is that workflows may be run across multiple, heterogeneous resources and the metrics should account for this. Our workflow metrics enable comparisons of performance regardless of the speed of the processors. For example, workflow parallel speedup will not change if the processors the workflow runs are upgraded; both the combined CPU hours and makespan will decrease. However, if parallelism is increased, even by using slower processors, workflow parallel speedup will improve. This is desired, since workflow parallel speedup is a measure of parallelism, not raw performance, which is measured by our computational

metrics. Running an application across multiple resources is often a way to improve performance and our workflow metrics capture this improvement.

These metrics, when considered together, can answer the questions posed previously. Task, flop, and file I/O numbers give a rounded picture of the workflow and computational demands of a scientific workflow calculation, and can be compared between different applications. Our proposed efficiency metrics portray an overall picture of the performance of a scientific workflow. The runtime of a workflow is affected by provisioned resources, the ability to schedule jobs to available resources, and the runtime of the individual jobs, and all these elements are reflected in our proposed metrics. These proposed metrics also reveal the balance necessary in getting good performance from scientific workflow applications. Provisioning additional cores may increase parallel speedup, but only if the workflow tools are able to schedule to the new cores so resource utilization remains high. Jobs can be clustered so there are more tasks per job, but this could decrease potential parallelism. To efficiently execute a scientific workflow, all these factors must be considered and carefully balanced.

#### 4. Performance analysis for CyberShake

We present a summary of the CyberShake values for the proposed metrics in Table 2 (task level), Table 3 (workflow level), and Table 4 (application level). These metrics reveal the size of the CyberShake computation and identify potential areas of improvement. CyberShake is clearly a massive application, both computationally and from a workflow perspective. Task-level metrics were gathered using *ioprof*, a wrapper that uses *strace* to capture file reads and writes to measure file I/O, *pprof*, a wrapper that uses *ptrace* to measure memory and CPU usage, and the PAPI toolkit. Workflow- and application-level metrics were gathered using the NetLogger toolkit (Tierney and Gunter, 2003), which mines runtime data generated by Pegasus for each task and job, and *gensim*, a Pegasus tool that determines overhead by parsing Condor logs.

Examining the job-level metrics, we see that multiple task types have significant I/O demands. This is confirmed in the application-level velocities of 113.9 GFLOP/s and 11 GB/s of I/O. Modification of the code to reduce I/O could result in better performance. We see that seismogram

**Table 3.** CyberShake workflow-level metrics, by workflow type.

Metric	SGT workflow	Post-processing workflow
Makespan	27.77 hrs.	15.08 hrs.
Workflow flop count	153.4 TFLOP	2.07 PFLOP
FLOPs	4.2 MFLOP/s	649 MFLOP/s
Workflow file I/O	236 GB	211 TB
File I/O per second	6.5 KB/s	4.0 GB/s
Compute time	10,005 CPU hrs	1444 CPU- hrs
Number of tasks	21	839,130
Tasks per second	<<1	15.5
Delay per job	606 sec	1482 sec
Workflow parallel speedup	360	96

SGT: strain Green's Tensor, FLOP: floating-point operation, I/O: input/output, CPU: central processing unit

**Table 4.** CyberShake application-level metrics.

Metric	CyberShake application
Makespan	1206.9 hours
Application flop count	494.9 PFLOP
FLOPs	113.9 GFLOP/s
Application file I/O	46.0 PB
File I/O per second	11.1 GB/s
Application compute time	2.59 million CPU hours
Total number of tasks	189 million
Total number of jobs	3,893,665
Tasks per second	43
Failed jobs	283 (0.007%)
Overall delay per job	1483
Application parallel speedup	2148
Resource utilization	27.8%

FLOP: floating-point operation, I/O: input/output, CPU: central processing unit

synthesis jobs read in, on average, 547 MB per job, but the input data is known to be 150–250 MB per job. This suggests that input data is being read multiple times, pointing to a possible area for optimization.

The workflow performance metrics help to characterize the two types of workflows, and reveal possible areas of improvement. The post-processing workflow is more demanding, having a higher flop count, higher FLOPs, and higher and faster file I/O. The low value for file I/O per second for the SGT workflow results from the majority of the SGT workflow being computation and messaging, with no checkpointing, and only brief file writes at the conclusion of the jobs. Having a delay per job of 1482 seconds for the post-processing workflow seems quite high. However, the character of the CyberShake post-processing workflow means that once SGT extraction jobs complete, many synthesis child jobs become eligible for execution, putting many jobs into the Condor queue. Since Condor can only schedule jobs to available glideins at a certain rate, it can take a long time for jobs in the back of the queue to execute. This, combined with the large number of synthesis jobs,

**Table 5.** CyberShake delay per job by delay type.

Delay Type	Percentage of delay
Submission delay	24.3
Queuing delay	74.3
Remote delay	1.4

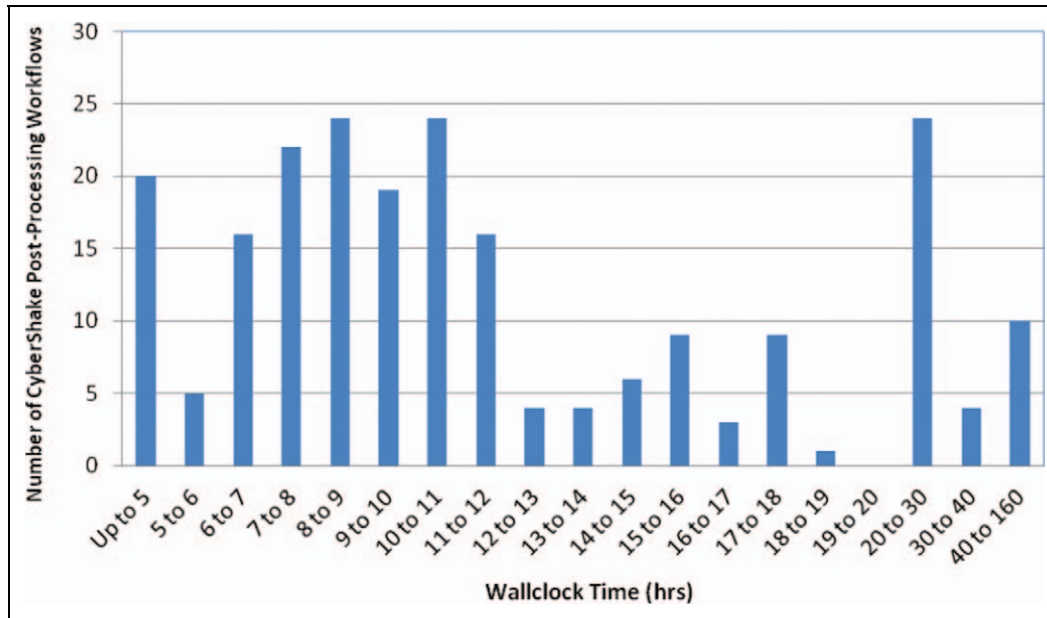
inflates the delay per job. Knowing our delay per job was high, we broke our delays down further in Table 5. Confirming our theory, the majority of delay was due to slow submission; that is, the delay between the completion of the job's last parent and the submission of the job.

Average delay per job of 606 seconds for the SGT workflow largely reflects the wait time of the Ranger queues. It is difficult to draw conclusions from this, but wait times of 10 minutes per job seem reasonable. It is challenging to improve this value without advance reservations or preemption.

The workflow parallel speedup of the CyberShake post-processing workflow was 96. This means that the post-processing workflow executed 96 times faster than could be done on a single Ranger core. The post-processing was performed on 400 cores, so while 96 shows meaningful speedup, the large number of jobs and inherent parallelism in the post-processing workflow suggests the possibility of larger speedup. Clearly at times cores were sitting idle. This metric can be applied to MPI jobs as well. The SGT workflow had a workflow parallel speedup of 360. This is expected, as the SGT workflow includes both 400 core MPI jobs and jobs with lower degrees of parallelism. Efforts to increase parallelism should be focused on the post-processing workflow.

The CyberShake research goal was the calculation of a probabilistic seismic hazard map requiring computation of more than 220 seismic hazard curves. The application-level metrics allow us to see the scope of the entire CyberShake run that accomplished this goal, as well as the overall degree of parallelization. Using a grid software stack including Pegasus and Condor, CyberShake executed its 189 million tasks at a rate of 43 tasks per second. As previously mentioned in Section 2.2, Pegasus provides the ability to bundle multiple tasks into a single job, so that the Condor scheduler sees fewer jobs than there are tasks, reducing load and overhead (scheduling and transferring jobs over the network). Hence, the number of jobs is significantly lower than the number of tasks. We also see that the failure rate is very low and easily managed by Condor's retry capability.

The application parallel speedup value of 2148 in the post-processing shows significant speedup, even over the workflow parallel speedup values. This is because not only is there parallelism within each workflow, but the application is also running entire workflows in parallel; specifically, multiple SGT and post-processing workflows simultaneously. The improvement over the workflow parallel speedup suggests that CyberShake is able to run workflows in parallel effectively.



**Figure 7.** Number of CyberShake post-processing workflows that ran in a given time.

In the post-processing workflow, CyberShake acquires glideins using Corral. This is done separately from job scheduling to glideins. As a result, there is the possibility that glideins could sit idle. Resource utilization of 27.8% suggests that this is occurring to some extent. Since workflow delay is quite high, this suggests that the scheduler is unable to schedule efficiently to available resources. These metrics indicate that while the Condor DAGManager continues to submit workflow jobs into the Condor queue, short-running jobs are finishing more quickly than the workflow stack can schedule new jobs, meaning glideins are sitting idle. Adjusting workflow parameters, such as cluster sizes and scheduler parameters, could improve the resource utilization and delay per job. Increasing the number of tasks per job would lengthen the runtime of each job and reduce the number of jobs, thereby reducing the load on the job scheduler and decreasing workflow overhead. Modifying scheduler parameters to increase scheduling frequency or improve scheduling efficiency could also decrease delay per job and increase resource utilization. Alternatively, employing a dynamic provisioning system for glideins could enable resources to scale with demand. For future CyberShake runs we plan to use CorralWMS (<http://pegasus.isi.edu/corralwms>), an integrated system merging Corral and GlideinWMS (Sfiligoi, 2008) that supports dynamic resource provisioning, enabling us to modify the size of the glidein pool to meet demand.

Figure 7 shows the number of post-processing workflows that ran in a certain amount of time. We can see that there is variability, and the high-runtime outliers are the result of system downtime. The large number of workflows running under 10 hours suggests that it is entirely possible to substantially improve the average post-processing runtime of 16.8 hours.

These metrics, in concert, quantify the scale and guide optimization efforts for our example application. Rather than acquiring more cores, effort should be made to increase parallelism by more efficiently using the available cores. Perhaps longer-running jobs would enable the scheduler to keep available resources busy. Another CyberShake run is planned for late 2010. These metrics are guiding optimization and will provide for easy measurement of improvement after optimized application runs.

## 5. Related work

There has been some investigation in the area of application metrics. HPC metrics are generally well established, with FLOPs per second (FLOP/s), millions of instructions per second (MIPS), and I/O per second all common measures. HTC metrics may include performance over longer time frames, such as operations per month or year, or application-specific metrics, such as simulations per year (Raman et al., 1998). Such domain-specific metrics limit the ability to compare performance between different applications. Raicu et al. (2008) suggest the use of typical HPC metrics such as FLOP/s, tasks/sec, or I/O rates for many-task workflow applications, which they define as including HPC and HTC jobs. While these are useful, they overlook some of the complexity that comes in running scientific workflows, such as task scheduling and multi-level parallelism.

In the area of workflow metrics, Ostermann et al. (2008) and Stratan et al. (2008) have defined a series of metrics designed to compare performance of grid workflow execution for serial jobs, including a detailed analysis of overhead (Nerieri et al., 2006; Prodan and Fahringer, 2008). These metrics include speedup compared to serial execution, total overhead, head-node consumption, and internal

stability. Some of the values needed to compute these metrics, such as critical path and overheads, may vary with each execution of the workflow and can be difficult to determine for a large heterogeneous workflow.

## 6. Conclusion

One of the challenges of large-scale heterogeneous scientific workflow applications is to quantify their scale and efficiency. Workflow metrics extend beyond scientific workflow applications to the workflow tools themselves. The same application could be run on multiple workflow middleware packages and these metrics used to compare performance.

Having well-defined workflow metrics enables developers to compare successive iterations of a computation. These metrics can be used to identify possible areas of optimization and measure the improvement as changes are made to the scientific codes as well as the workflow tools. Consideration of these metrics in the context of CyberShake has revealed a number of areas of potential improvement. As modifications are made to CyberShake, these metrics provide a means to measure and document increased performance.

In addition, agreeing on a set of metrics provides a basis for comparison between applications. In this way workflow and computational demands can be quantified on an even basis, even for workflows that differ in character. As an example scientific workflow calculation, CyberShake can benefit from analyzing these metrics and applying their results to future iterations. Other applications could benefit as well, and we invite other scientific workflows to describe their workflow application performance measures using these metrics as a basis for comparison and analysis.

We hope this examination of scientific workflow metrics will encourage formalization of performance statistics and greater understanding of the needs and scale of scientific workflow applications.

## Funding

This work was partially supported by: TACC (grant number TG-MCA03S012; SCEC PetaScale Research: An Earthquake System Science Approach to Physics-based Seismic Hazard Research) and utilized TACC's Ranger cluster; in part by the Center for HPCCs at the USC; this work was also supported by NSF (grant numbers OCI-0722019 and OCI-0749313). The SCEC is funded by the National Science Foundation (NSF; Cooperative Agreement EAR-0106924) and USGS (Cooperative Agreement 02HQAG0008). The SCEC contribution number for this paper is 1484.

## Conflict of interest statement

None declared.

## References

Callaghan S, et al. (2008) Reducing time-to-solution using distributed high-throughput mega-workflows - experiences from

- SCEC CyberShake. In: *Proceedings of the Fourth IEEE International Conference on e-Science (e-Science 2008)*, Indianapolis, Indiana, USA.
- Callaghan S, et al. Scaling up workflow-based applications. *J Comput Syst Sci* 76: 428-446.
- Deelman E, et al. (2005) Pegasus: a framework for mapping complex scientific workflows onto distributed systems. *Sci Program J* 13: 219-237.
- Dongarra J, London K, Moore S, Mucci P and Terpstra D (2001) Using PAPI for hardware performance monitoring on linux systems. In: *Proceedings of the Conference on Linux Clusters: The HPC Revolution*, Linux Clusters Institute, Urbana, Illinois, 25-27 June.
- Graves R, et al. (2008) Physics based probabilistic seismic hazard calculations for Southern California. In: *Proceedings of the 14th World Conference on Earthquake Engineering*, Beijing, China, 12-17 October.
- Nerieri F, et al. (2006) Overhead analysis of grid workflow applications. In: *Proceedings of the 7th IEEE/ACM International Conference on Grid Computing*, pp. 17-24.
- Ostermann S, et al. (2008) On the characteristics of grid workflows. In: *Proceedings of the CoreGRID Workshop on Integrated Research in Grid Computing (CGIW'08)*, pp. 431-442.
- Prodan R and Fahringer T (2008) Overhead analysis of scientific workflows in grid environments. *IEEE Trans Parallel Distrib Syst* 19: 378-393.
- Raicu I, Foster T and Zhao Y (2008) Many-task computing for grids and supercomputers. *Workshop on Many-Task Computing for Grids and Supercomputers 2008*, pp. 1-11.
- Raman R, Livny M and Solomon M (1998) Matchmaking: distributed resource management for high throughput computing. In: *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC-7 '98)*, pp. 140-146.
- Sfiligoi I glideinWMS—a generic pilot-based workload management system. *J Phys Conf Ser*.
- Stratan C, Iosup A and Epema DHJ (2008) A performance study of grid workflow engines. In: *Proceedings of the 9th IEEE/ACM International Conference on Grid Computing*, pp. 25-32.
- Taylor I, Deelman E, Gannon D and Shields M (eds) (2006) *Workflows in e-Science*. New York: Springer.
- Tierney B and Gunter D (2003) NetLogger: a toolkit for distributed system performance tuning and debugging. In: *Proceedings of the 8th IFIP/IEEE International Symposium on Integrated Network Management*.
- Zhao L, Chen P and Jordan TH (2006) Strain Green's tensors, reciprocity, and their applications to seismic source and structure studies. *Bull Seismol Soc Am* 96: 1753-1763.

## Author's Biographies

Scott Callaghan graduated with his BS and MS degrees in Computer Science in 2004 and 2007, respectively, from USC in Los Angeles. He is a research programmer at SCEC. His research interests include high-performance computing, high-throughput computing, scientific workflows, and seismic hazard analysis.

*Philip Maechling* is the Information Technology Architect for the SCEC at the USC where he manages SCEC's scientific computing projects, including the SCEC Community Modeling Environment (SCEC/CME), an earthquake system science research collaboration, and the SCEC Collaboratory for the Study of Earthquake Predictability (CSEP), an international earthquake forecast testing facility.

*Patrick Small* graduated with his BS degree in Computer Science in 1996 from the University of British Columbia (UBC) in Vancouver, BC. He is a research programmer at SCEC. His research interests include high-performance computing and seismic velocity model development.

*Kevin Milner* graduated with a BS in Computer Science in 2007 from USC in Los Angeles. Initially a participant in the SCEC's undergraduate research program for two years, he is now a research programmer with SCEC. His research interests include seismic hazard analysis, operational earthquake forecasting, high-throughput computing, and scientific workflows.

*Gideon Juve* received BS and MS degrees in Computer Science from the USC in 2004 and 2008. He is currently a PhD. candidate in Computer Science at USC. His research interests include computational workflows and scientific computing.

*Thomas H Jordan* is Director of SCEC and University Professor at USC. His research is focused on system-level models of earthquake processes, earthquake forecasting, and continental dynamics. He is a member of the California Earthquake Prediction Evaluation Council, the Governing Board of the US National Research Council, and the Board of Directors of the Seismological Society of America. He received his PhD from the California Institute of Technology in 1972. He taught at Princeton University and the Scripps Institution of Oceanography before joining the Massachusetts Institute of Technology (MIT) as the Robert R. Shrock Professor in 1984. He served as the head of the MIT's Department of Earth, Atmospheric and Planetary Sciences for the decade 1988–1998. In 2000, he moved from the MIT to the USC, and in 2004, he was appointed as a USC University Professor. He has been awarded the Macelwane and Lehmann Medals of the American Geophysical Union and the Woollard Award of the Geological Society of America. He is a member of the National Academy of Sciences, the American Academy of Arts and Sciences, and the American Philosophical Society.

*Ewa Deelman* is a Research Associate Professor at the USC Computer Science Department and a Project Leader at the USC Information Sciences Institute (ISI). Her research

interests include the design and exploration of collaborative, distributed scientific environments, with particular emphasis on workflow management, as well as the management of large amounts of data and metadata. At ISI, she is leading the Pegasus project, which designs and implements workflow mapping techniques for large-scale workflows running in distributed environments. She received her PhD from Rensselaer Polytechnic Institute in Computer Science in 1997 in the area of parallel discrete event simulation.

*Gaurang Mehta* is a Research Programmer at ISI at the USC. He received his MS in Electrical Engineering from USC and his BE in Electronics Engineering from Mumbai University, India. He is currently the co-developer of the Pegasus workflow management system and a lead on enabling several bioinformatics projects to use Pegasus.

*Karan Vahi* is a Research Programmer at USC ISI. He is a member of the Advanced Systems Division at ISI and works on the Pegasus Project. He received a MS in Computer Science from the USC and a BE in Computer Engineering from Thapar University, India. He has been associated with the Pegasus project since its inception. He is currently the lead developer on it and works closely with the user community to drive its development.

*Dan Gunter* graduated with a MS in Computer Science from San Francisco State University (SFSU) in 1999. He is an Associate Computer Scientist in the Computational Research Division at Lawrence Berkeley National Laboratory. His research interests include performance analysis of data-intensive and distributed scientific computing.

*Keith Beattie* is a computer systems engineer at Lawrence Berkeley National Laboratory with over 15 years combined industry and research experience developing software. His interests are in distributed systems and effective software development and release management practices. He holds a MS in Computer Science and a BA in Mathematics from SFSU.

*Christopher Brooks* is Associate Dean of Sciences and Associate Professor of Computer Science at the University of San Francisco. He received his BA/JBA from the University of Wisconsin, his MS from SFSU in 1997, and his PhD from the University of Michigan in 2002. He is the director of Community Connections, a service-learning project within the computer science program. His interests are in the union of the sets of artificial intelligence and distributed systems: multiagent systems, peer-to-peer systems, machine learning, intelligently dealing with Web data, and electronic commerce.