Future Generation Computer Systems 29 (2013) 682–692

Contents lists available at SciVerse ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

Characterizing and profiling scientific workflows

Gideon Juve [*], Ann Chervenak, Ewa Deelman, Shishir Bharathi, Gaurang Mehta, Karan Vahi

University of Southern California Information Sciences Institute, 4676 Admiralty Way, Marina del Rey, CA 90292, United States

ARTICLE INFO

Article history:
Received 16 December 2011
Received in revised form
20 July 2012
Accepted 6 August 2012
Available online 27 September 2012

Keywords:
Scientific workflows
Profiling
Performance
Measurement

ABSTRACT

Researchers working on the planning, scheduling, and execution of scientific workflows need access to a wide variety of scientific workflows to evaluate the performance of their implementations. This paper provides a characterization of workflows from six diverse scientific applications, including astronomy, bioinformatics, earthquake science, and gravitational-wave physics. The characterization is based on novel workflow profiling tools that provide detailed information about the various computational tasks that are present in the workflow. This information includes I/O, memory and computational characteristics. Although the workflows are diverse, there is evidence that each workflow has a job type that consumes the most amount of runtime. The study also uncovered inefficiency in a workflow component implementation, where the component was re-reading the same data multiple times.

1. Introduction

Computational scientists are running increasingly complex, data-intensive simulations and analyses. Many scientific communities use workflow technologies to manage this complexity [1]. Workflow technologies are responsible for scheduling computational tasks on distributed resources, for managing dependencies among tasks, and for staging datasets into and out of execution sites.

As scientific workflows grow in complexity and importance, the designers of workflow management systems need a deeper and broader understanding of what workflows require and how they behave in order to drive improvements in algorithms for provisioning resources, scheduling computational jobs, and managing data. To date, the community has lacked detailed knowledge of a range of scientific workflows because few workflow applications have been available for general use, since many workflow users are reluctant to release their code and data. One exception is the Montage astronomy workflow [2], which has been used extensively to evaluate workflow algorithms and systems. While Montage has proven to be a very useful resource for the community [3–11], workflow systems intended for general use should not be designed and evaluated based on the characteristics of a single workflow.

In this paper, we characterize six scientific workflows and provide a profile for a typical execution of each workflow. In characterizing the execution profile for each workflow, we present such parameters as the number of jobs of each type and the I/O, memory, and CPU requirements of each job type. The workflows that we characterize are taken from diverse application domains such as astronomy, biology, gravitational physics and earthquake science. Together, they provide a broad overview of the types of applications that currently use workflow technologies to manage complex analyses. They also represent different types of workflows from small to large-scale, and include I/O-intensive, CPU-intensive, and memory-bound codes.

The goal of these workflow characterizations is to provide a community resource that will allow the designers and users of workflow systems to base their evaluation of algorithms and technologies on the characteristics of a broader range of workflows. These characterizations can be used by the research community even when application code and data are not available to develop synthetic workflows, benchmarks, and simulations for evaluating workflow management systems.

Prior work [12,13] has focused on workflow patterns and workflow system taxonomies, respectively. In this paper, we focus on workflow characterization from the point of view of the performance of the individual workflow components and overall workflows. The goal is to obtain detailed task-level performance metrics similar to those described in [14,15]. In addition to being useful for understanding and simulating workflows to improve workflow management systems, this data has a number of other uses. Many existing workflow systems provide support for capturing provenance [16–18]. Profile data could be stored along with provenance to provide a more complete history of the execution of a workflow. The information could also be used to improve workflow scheduling. Many scheduling algorithms require runtime and data estimates in order to

[*] Corresponding author. Tel.: +1 9254132099.
E-mail addresses: juve@usc.edu, gideon@isi.edu (G. Juve), annc@isi.edu (A. Chervenak), deelman@isi.edu (E. Deelman), shishir@isi.edu (S. Bharathi), gmehta@isi.edu (G. Mehta), vahi@isi.edu (K. Vahi).

0167-739X/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2012.08.015

optimize workflow execution [19–22]. Profile data could be used to create a performance model of the workflow that is able to generate these estimates. Similarly, profile data could be used to guide resource provisioning algorithms that require estimates of resource usage [23–26].

This paper is organized as follows. Section 2 provides an overview of the scientific workflows that we characterize. In Section 3, we discuss the profiling tools used to determine a typical execution profile for each workflow. In Section 4, we present detailed characterizations of the scientific workflows described in Section 2. We present an execution profile for each workflow running at a typical scale and managed by the Pegasus workflow management system [27–29]. In these profiles, we characterize the I/O, memory, and computational requirements of each job type in the workflow. Finally, we discuss related work and outline directions for future work.

2. Overview of scientific workflows

We now briefly describe the scientific workflows that we have characterized.

Montage: Montage [2] was created by the NASA/IPAC Infrared Science Archive as an open source toolkit that can be used to generate custom mosaics of the sky using input images in the Flexible Image Transport System (FITS) format. During the production of the final mosaic, the geometry of the output image is calculated from the input images. The inputs are then re-projected to have the same spatial scale and rotation, the background emissions in the images are corrected to have a uniform level, and the re-projected, corrected images are co-added to form the output mosaic. The Montage application has been represented as a workflow that can be executed in Grid environments such as the TeraGrid [30].

CyberShake: The CyberShake workflow [31] is used by the Southern California Earthquake Center (SCEC) [32] to characterize earthquake hazards using the Probabilistic Seismic Hazard Analysis (PSHA) technique. Given a region of interest, an MPI based finite difference simulation is performed to generate Strain Green Tensors (SGTs). From the SGT data, synthetic seismograms are calculated for each of a series of predicted ruptures. Once this is done, spectral acceleration and probabilistic hazard curves are calculated from the seismograms to characterize the seismic hazard. CyberShake workflows composed of more than 800,000 jobs have been executed using the Pegasus workflow management system on the TeraGrid [33,34].

Broadband: Broadband is a computational platform used by the Southern California Earthquake Center [32]. The objective of Broadband is to integrate a collection of motion simulation codes and calculations to produce research results of value to earthquake engineers. These codes are composed into a workflow that simulates the impact of one or more earthquakes on one of several recording stations. Researchers can use the Broadband platform to combine low frequency (less than 1.0 Hz) deterministic seismograms with high frequency ($\sim$10 Hz) stochastic seismograms and calculate various ground motion intensity measures (spectral acceleration, peak ground acceleration and peak ground velocity) for building design analysis.

Epigenomics: The USC Epigenome Center [35] is currently involved in mapping the epigenetic state of human cells on a genome-wide scale. The Epigenomics workflow is essentially a data processing pipeline that uses the Pegasus workflow management system to automate the execution of the various genome sequencing operations. DNA sequence data generated by the Illumina-Solexa [36] Genetic Analyzer system is split into several chunks that can be operated on in parallel. The data in each chunk is converted into a file format that can be used by the Maq software that maps short DNA sequencing reads [37,38]. From there, the sequences are filtered to remove noisy and contaminating segments and mapped into the correct location in a reference genome, Finally, a global map of the aligned sequences is generated and the sequence density at each position in the genome is calculated. This workflow is being used by the Epigenome Center in the processing of production DNA methylation and histone modification data.

LIGO Inspiral Analysis: The Laser Interferometer Gravitational Wave Observatory (LIGO) [39,40] is attempting to detect gravitational waves produced by various events in the universe as per Einstein's theory of general relativity. The LIGO Inspiral Analysis workflow [41] is used to analyze the data obtained from the coalescing of compact binary systems such as binary neutron stars and black holes. The time–frequency data from each of the three LIGO detectors is split into smaller blocks for analysis. For each block, the workflow generates a subset of waveforms belonging to the parameter space and computes the matched filter output. If a true inspiral has been detected, a trigger is generated that can be checked with triggers for the other detectors. Several additional consistency tests may also be added to the workflow.

SIPHT: The bioinformatics project at Harvard University is conducting a wide search for small, untranslated RNAs (sRNAs) that regulate processes such as secretion and virulence in bacteria. The sRNA Identification Protocol using High-throughput Technology (SIPHT) program [42] uses a workflow to automate the search for sRNA encoding-genes for all bacterial replicons in the National Center for Biotechnology Information (NCBI) database. The kingdom-wide prediction and annotation of sRNA encoding genes involves a variety of programs that are executed in the proper order using Condor DAGMan's [43,44] capabilities. These involve the prediction of Rho-independent transcriptional terminators, BLAST (Basic Local Alignment Search Tools) comparisons of the intergenetic regions of different replicons and the annotations of any sRNAs that are found.

3. Characterization of workflows

We have developed a set of workflow profiling tools called wfprof[1] to collect and summarize performance metrics for workflow applications. Wfprof is composed of two tools – ioprof and pprof – that wrap the jobs in a workflow and record trace data as they run. Ioprof collects data about process I/O, and pprof collects data about process runtimes, memory usage, and CPU utilization. We use both ioprof and pprof because the runtime overhead of ioprof can impact the statistics collected by pprof.

The ioprof tool characterizes I/O activity in the workflow using the strace (system call trace) utility [45] to intercept system calls made by a job. The system calls are mined for data about operations performed on file descriptors. The calls of interest include open(), close(), read(), write(), and others. The arguments and return values from these system calls are analyzed to determine which files were accessed by the job, how they were accessed, and how much I/O was performed. This process involves some overhead, but since the statistics collected are independent of time, they should not be impacted by this overhead. In addition to file I/O, this method also captures I/O performed on standard streams (stdin, stdout, stderr), pipes, and sockets. It is important to note that file I/O may not be a good estimate of actual disk I/O because the file system cache may absorb some operations that would otherwise result in disk activity. Therefore, the reported numbers reflect the amount of data the workflows requested, not necessarily the amount of work performed by the storage system.

[1] http://pegasus.isi.edu/wfprof.

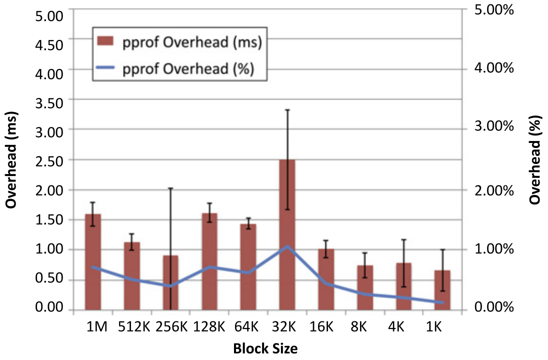

Fig. 1. Overhead of pprof for a task that reads and writes 1GB of data using various block sizes. The figure shows the absolute overhead in milliseconds and the overhead as a percentage of the runtime without profiling. This figure shows that the pprof overhead is very low at 0.5–2.5 ms per task, or less than 1% of the total runtime without profiling.

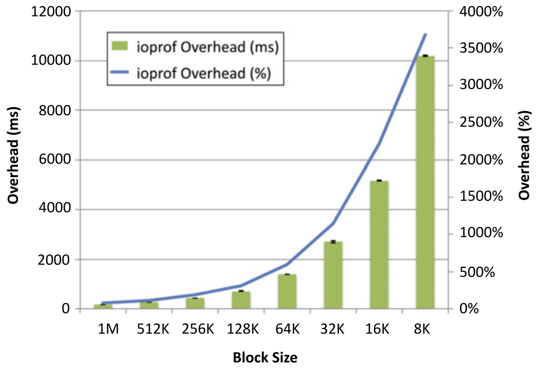

Fig. 2. Overhead of ioprof for a task that reads and writes 1GB of data using various block sizes. This figure shows the absolute overhead in milliseconds and the overhead as a percentage of the runtime without profiling. The block size determines the number of system calls required to complete the operation. Because ioprof intercepts system calls, its overhead is determined by the number of system calls.

The pprof tool characterizes the process runtime, memory usage and CPU utilization of a workflow using the Linux ptrace (process trace) API [46,47] to analyze lifecycle events for the processes invoked by a job. The kernel notifies pprof whenever a process starts or finishes, and pprof records data about the process such as the start and stop times, the peak RSS (resident set size, or physical memory usage), time spent in user mode (utime), and time spent in kernel mode (stime). All these values, with the exception of start and stop time, are maintained by the kernel for every process. Unlike ioprof, pprof does not intercept every system call, so the overhead is much lower.

To demonstrate how profiling affects the runtime of a typical workflow task, we ran several experiments using a simple benchmark program that reads 1 GB of data from an infinite source (/dev/zero) and writes it to an infinite sink (/dev/null) using various block sizes. This program was executed without profiling, with pprof profiling, and with ioprof profiling. We ran 100 experiments for each of 10 different block sizes using all three profiling configurations on a quad core, 2.4 GHz Intel Core 2 processor running Fedora 10 with a 2.6.27 Linux kernel. These experiments are included for illustration purposes and are not meant to be a comprehensive evaluation of the profiling tools. The results of these experiments are shown in Figs. 1 and 2.

Fig. 1 shows the overhead of pprof in milliseconds calculated by subtracting the mean runtime of the experiments with no profiling from the experiments using pprof, and the overhead of pprof as a percentage of the runtime without profiling. For this benchmark program the results show that the overhead of pprof is between 0.5 and 2.5 ms, or less than 1% of the unprofiled runtime. In addition, the pprof overhead is nearly uniform across different block sizes. The small variation in overhead across different block sizes is likely a result of kernel-level implementation details.

Fig. 2 shows the overhead for ioprof using the same calculation for overhead as was used for pprof. Note that for ioprof we omit results for 4 and 1 K block sizes because of the long runtime of the experiments. Unlike pprof, which has a very low, fairly uniform overhead, ioprof's overhead is much larger and depends on the number of system calls made by the benchmark program, which is determined by the block size used in the experiment. Each time the block size is halved, the number of system calls doubles and, as a result, the overhead of ioprof doubles.

It should be noted that the metrics measured by pprof and ioprof are not impacted by these profiling overheads. In the case of pprof, although it adds ∼1–2 ms to the runtime of a workflow task, this overhead is not included in the runtime it is measuring, which remains unchanged. In the case of ioprof, although the profiling does add significantly to the runtime of the task, ioprof only measures I/O, which is not affected by the runtime. Both tools do have an impact on the runtime of the workflow, however, which may be significant depending on the number of tasks in the workflow and the amount of I/O performed by the workflow.

To characterize a workflow, we run it twice: once to collect I/O data using ioprof and once to collect process data using pprof. These runs produce workflow traces that can be examined for errors and unusual behavior, analyzed for optimization opportunities, fed into a simulation, or mined to collect summary statistics for a workflow profile. In this paper, we report profile data in Section 4 and provide some examples of how this data can be used.

To create a workflow profile, we process the traces from ioprof and pprof to calculate the mean and standard deviation for five statistics for each job type: I/O read, I/O write, runtime, peak physical memory usage, and CPU utilization. I/O read and write are calculated by adding up all the I/O performed by all processes invoked by a job. Runtime is calculated by subtracting the start time of the first process invoked by the job from the stop time of the last process. Peak memory usage is calculated from the total peak memory usage of all processes invoked by the job. Since a single job may invoke many different processes concurrently, care is taken to ensure that the peak memory measurements of concurrent processes are added together and the maximum value over the lifetime of the job is reported. Because the peak memory usage of a process may not be maintained for the entire lifetime of the process, there may be some error when adding these values together. As a result, the peak memory reported represents an upper bound on memory usage rather than the actual peak. CPU utilization is calculated by dividing the amount of time each process spends scheduled in user mode (utime) and kernel mode (stime) by the total runtime of the process. In the case where a job consists of several processes running concurrently, the total utime + stime of all processes is divided by the wall time of the job. This computation results in the percentage of time the process spends running on the CPU when it is not blocked waiting for I/O operations to complete. Note that for CPU-intensive concurrent processes running on multicore nodes, the percentage may be greater than 100%.

4. Characterization of example workflows

In this section, we provide characterizations of the six workflows mentioned in Section 2. We present profiles that were obtained from actual executions of the workflows on the Grid. Each workflow instance was chosen to be a typical size for the given application.

Table 1
Experimental conditions for profiling and normal executions. The table shows the execution systems for each application and shows the overhead associated with the execution of the workflow with and without profiling. For CyberShake we do not have a complete set of results.

Workflow	Site	Nodes	Cores	CPU (GHz)	Memory (GB)	File system	ioprof	pprof	Normal
Montage	Amazon EC2	1	8	Xeon @ 2.33	7.5	ext3	1:11:20	0:57:36	0:55:28
CyberShake	TACC Ranger	36	16	Opteron @ 2.3	32	Lustre	16:35:00		N/A
		15					–	12:20:00	–
Broadband	Amazon EC2	1	8	Xeon @ 2.33	7.5	ext3	1:31:24	1:21:16	1:20:03
Epigenome	Amazon EC2	1	8	Xeon @ 2.33	7.5	ext3	1:11:11	1:08:01	1:07:40
LIGO	Syracuse SUGAR	80	4	Xeon @ 2.50	7.5	NFS	1:48:11	1:47:38	1:41:13
SIPHT	UW Newbio	13	8	Xeon @ 3.16	1	ext3	1:33:24	1:19:37	1:10:53

Table 2
Overview of workflow execution profiles. This table summarizes profile information at the workflow level.

Workflow	Jobs	CPU (h)	I/O read (GB)	I/O write (GB)	Peak memory (MB)	CPU utilization (%)
Montage	10 429	4.93	146.01	49.93	16.77	31.04
CyberShake	815 823	9192.45	217 369.37	920.93	1870.38	90.89
Broadband	770	9.48	1171.73	175.41	942.32	88.85
Epigenome	529	7.45	24.14	5.36	197.47	95.91
LIGO	2 041	51.17	209.36	0.05	968.68	90.50
SIPHT	31	1.23	1.70	1.48	116.38	93.37

A summary of the execution profile statistics for all the workflows we examined is shown in Tables 1 and 2. The data show a wide range of workflow characteristics. The workflows that we characterize include small workflows (SIPHT), CPU-bound workflows (Epigenome), I/O-bound workflows (Montage), data-intensive workflows (CyberShake, Broadband), workflows with large memory requirements (CyberShake, Broadband, LIGO), and workflows with large resource requirements (CyberShake, LIGO). We will examine each of these applications in more detail in the coming sections.

For each workflow, Table 1 shows the two profiling runs (ioprof and pprof) and a third run without any profiling enabled (normal) for comparison. These elapsed times are shown in units of hours:minutes:seconds. The differences between the elapsed time of the profiling runs and the normal run are small and fairly consistent; I/O profiling has higher overhead.

Table 1 also shows the execution environments that were used to collect the data. Several of the workflows (Epigenome, Montage, Broadband) were run on Amazon's cloud computing platform EC2. The other workflows (SIPHT, CyberShake, LIGO) were run on the clusters used by the developers of the application. The workflows run on EC2 all used a local disk with the ext3 file system for sharing data between jobs. Those runs used only a single node, so no inter-node transfers were required. SIPHT also used a local file system; however, the input and output files were transferred to the worker nodes for each job. The remaining workflows used network file systems to share data among workers (Lustre for CyberShake and NFS for LIGO). We do not present the "Normal" runtime for the CyberShake workflow, because the data available to us were obtained from a workflow execution with several failures and breaks in time in the workflow and thus are not representative.

Table 2 shows the execution profiles for the six workflows, including the number of jobs in the workflow, the total CPU hours consumed in running the workflow, the amount of I/O read and written, the peak memory usage and the average CPU utilization of the workflow. The CyberShake workflow is the largest of those profiled, consisting of 815,823 jobs and consuming 9192 CPU hours; the workflow reads 217 terabytes of data and writes 920 GB. The smallest workflow profiled is SIPHT, which consists of 31 jobs, consumes 1.23 CPU hours and transfers less than 2 GB of read and write data.

A detailed comparison of the runtimes of individual jobs reported by pprof show that the profiling overhead of pprof was very low for these workflows (<1 ms per task) and that the difference between the normal and pprof elapsed runtimes is primarily due to the extra data transfers required to retrieve the trace data from the remote system. Recall from the last section that pprof is used for runtime characteristics and ioprof is used only for I/O characteristics, so extra ioprof overhead does not affect the runtime data. In addition, some workflow characteristics, such as number of jobs, I/O read, I/O write, and peak memory are independent of the execution environment and the profiler overhead. Runtime and CPU utilization depend on the quantity and performance of resources used to execute the workflow and will vary from platform to platform.

4.1. Montage

In Fig. 3, we show a relatively small (20 node) Montage astronomy workflow for the purpose of illustrating the workflow's structure. The size of a Montage workflow depends on the number of images used in constructing the desired mosaic of the sky. The structure of the workflow changes to accommodate increases in the number of inputs, which corresponds to an increase in the number of computational jobs.

At the top level of the workflow shown in Fig. 3 are mProjectPP jobs, which reproject input images. The number of mProjectPP jobs is equal to the number of Flexible Image Transport System (FITS) input images processed. The outputs of these jobs are the reprojected image and an "area" image that consists of the fraction of the image that belongs in the final mosaic. These are then processed together in subsequent steps. At the next level of the workflow, mDiffFit jobs compute a difference for each pair of overlapping images. The number of mDiffFit jobs in the workflow therefore depends on how the input images overlap. The difference images are then fitted using a least squares algorithm by the mConcatFit job. The mConcatFit job is a computationally intensive data aggregation job. Next, the mBgModel job computes a correction to be applied to each image to obtain a good global fit. This background correction is applied to each individual image at the next level of the workflow by the mBackground jobs. The mConcatFit and mBgModel jobs are data aggregation and data

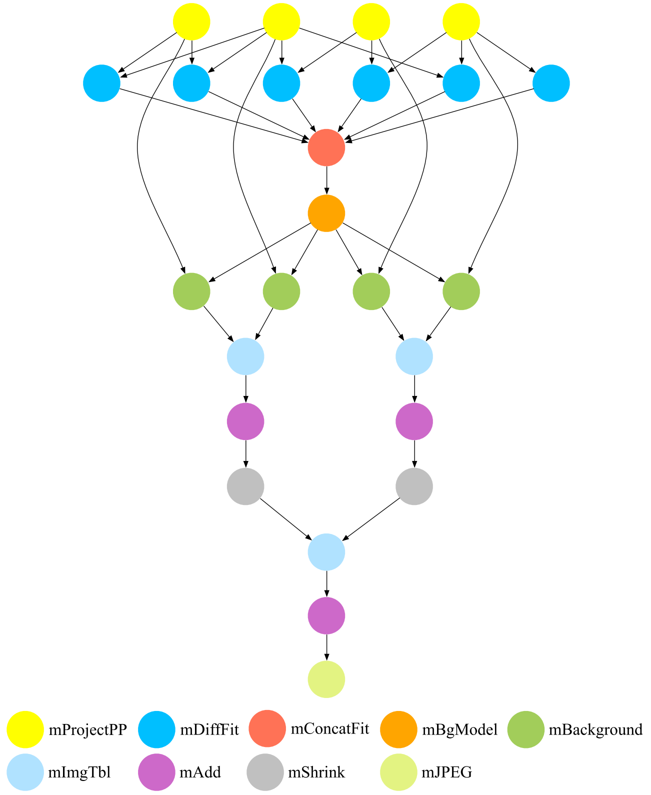

Fig. 3. Montage workflow. The circles represent the various computational workflow tasks. The arrows represent the data dependencies between the tasks. Different colors represent different job types.

partitioning jobs, respectively, but together they can be considered as a single data redistribution point. Note that there is not a lot of data being partitioned in this case. Rather, the same background correction is applied to all images. The mImgTbl job aggregates metadata from all the images and creates a table that may be used by other jobs in the workflow. The mAdd job combines all the reprojected images to generate the final mosaic in FITS format as well as an area image that may be used in further computation. The mAdd job is the most computationally intensive job in the workflow. Multiple levels of mImgTbl and mAdd jobs may be used in large workflows. Finally, the mShrink job reduces the size of the FITS image by averaging blocks of pixels and the shrunken image is converted to JPEG format by mJPEG.

In Table 3, we provide profiling data from the execution of a Montage workflow on the Grid that computes a mosaic for an 8 degree square region of the sky. Table 3 shows that the majority of runtime was spent in the 17 instances of the mAdd job, which performed the most I/O on average (1 GB of data read and 775 GB of data written per job) but had low CPU utilization. The table shows low CPU utilization for several job types in the workflow (mBackground, mImgtbl, mAdd, mShrink). This is because Montage jobs spend much of their time on I/O operations, as illustrated by their I/O and runtime measurements; for example, the mShrink jobs have a relatively short runtime, require a large amount of read I/O, and have low CPU utilization. The single mBgModel job had the longest mean job runtime, the highest CPU utilization (99.89%) and low I/O activity.

The table also shows that mAdd has high standard deviations for all measurements. This is because the same mAdd executable is used in two different ways in the workflow. The workflow has a two-level reduce structure that uses mAdd, where the second level has much different runtime characteristics than the first level. These two levels could be treated as two different job types to clarify the statistics.

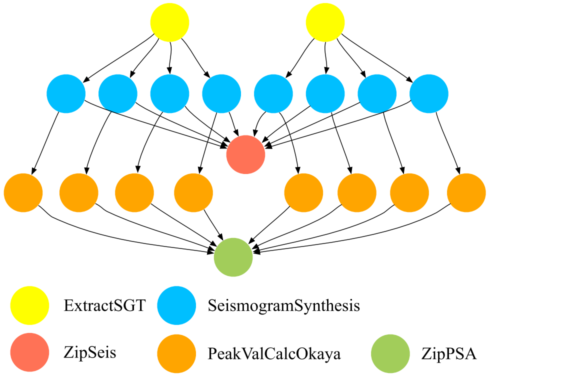

Fig. 4. CyberShake workflow. This workflow is highly parallel. The aggregation nodes in the workflow are placed to combine a large number of small data files prior to data transfer.

4.2. CyberShake

In Fig. 4, we show a small CyberShake earthquake hazard characterization workflow. While relatively simple in structure, this workflow can be used to perform significant amounts of computation on extremely large datasets. Strain Green Tensor (SGT) data generated from finite simulations is maintained in the form of large "master" SGT files for *x* and *y* dimensions. This master SGT data quantifies the relationship between motion at a site and motion throughout the region. In addition to SGT data, there is a collection of estimated future fault ruptures and variations of those ruptures. These datasets are combined to estimate the seismic hazard at the site. The ExtractSGT jobs in the workflow extract the SGTs pertaining to a given rupture from the master SGT files for the site. ExtractSGT jobs may therefore be considered data partitioning jobs.

Synthetic seismograms are generated for each rupture variation by the SeismogramSynthesis jobs. Peak intensity values, in particular the spectral acceleration, are calculated by the PeakValueCalcOkaya jobs for each synthetic seismogram. The resulting synthetic seismograms and peak intensities are collected and compressed by the ZipSeismograms and ZipPeakSA jobs to be staged out and archived. These jobs may be considered as simple data aggregation jobs.

Of the computational jobs, seismogram synthesis jobs are the most computationally intensive. However, when the workflow is executed on the Grid, due to the large sizes of the master SGT files, ExtractSGT jobs may also consume a lot of time on compute resources. Additionally, since a lot of data may be generated by the workflow, the ZipSeismograms and ZipPeakSA jobs may also consume a large amount of time in generating the compressed files to be staged out.

In Table 4, we provide profiling data from the execution of a large (more than 800,000 jobs) CyberShake workflow on the Ranger cluster at TACC. Note that in order to fit into the table, SeismogramSynthesis, PeakValueCalcOkaya and ZipSeismograms jobs are represented as SeisSynth, PVCOkaya and ZipSeis, respectively.

The vast majority of the runtime (>97%) of CyberShake is spent in SeismogramSynthesis jobs. This suggests that SeismogramSynthesis would be a good place to focus any code-level optimization efforts. In addition, Seismogram-Synthesis reads, on average, 547 MB per job; however, the total size of all the input files for SeismogramSyntheis is known to be on the order of 150–250 MB. That means some of the input data are being read multiple times, which suggests another opportunity for code-level optimization. Finally, SeismogramSynthesis also has very high memory requirements. On average it requires 817 MB of memory, but the profiling data shows that some instances require up to 1870 MB, which

Table 3
Example of a Montage execution.

Job	Count	Runtime		I/O read		I/O write		Peak memory		CPU util.	
		Mean (s)	Std. dev.	Mean (MB)	Std. dev.	Mean (MB)	Std. dev.	Mean (MB)	Std. dev.	Mean (%)	Std. dev.
mProjectPP	2102	1.73	0.09	2.05	0.07	8.09	0.31	11.81	0.32	86.96	0.03
mDiffFit	6172	0.66	0.56	16.56	0.53	0.64	0.46	5.76	0.67	28.39	0.16
mConcatFit	1	143.26	0.00	1.95	0.00	1.22	0.00	8.13	0.00	53.17	0.00
mBgModel	1	384.49	0.00	1.56	0.00	0.10	0.00	13.64	0.00	99.89	0.00
mBackground	2102	1.72	0.65	8.36	0.34	8.09	0.31	16.19	0.32	8.46	0.10
mImgtbl	17	2.78	1.37	1.55	0.38	0.12	0.03	8.06	0.34	3.48	0.03
mAdd	17	282.37	137.93	1102.57	302.84	775.45	196.44	16.04	1.75	8.48	0.11
mShrink	16	66.10	46.37	411.50	7.09	0.49	0.01	4.62	0.03	2.30	0.03
mJPEG	1	0.64	0.00	25.33	0.00	0.39	0.00	3.96	0.00	77.14	0.00

Table 4
Example of a CyberShake execution.

Job	Count	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	5 939	110.58	141.20	175.40	177.70	155.86	176.48	20.64	0.64	65.82	0.28
SeisSynth	404 864	79.47	70.86	547.16	324.92	0.02	0.00	817.59	483.51	92.01	0.08
ZipSeis	78	265.73	275.04	118.95	7.88	101.05	14.36	6.25	0.16	6.83	0.01
PVCOkaya	404 864	0.55	2.48	0.02	0.00	0.00	0.00	3.11	0.01	16.89	0.04
ZipPeakSA	78	195.80	237.99	1.07	0.07	2.26	0.15	6.16	0.16	2.89	0.01

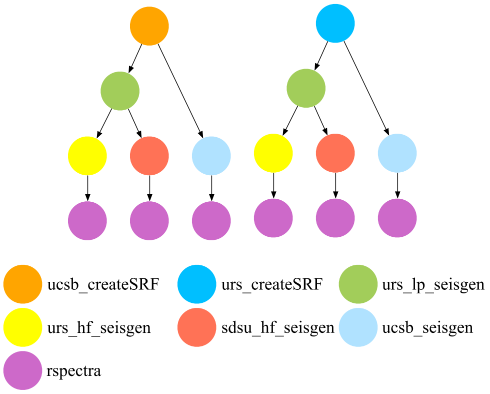

Fig. 5. Broadband workflow. The workflow simulates earthquakes using several different models and generates output data for comparison.

was not previously known. This has an impact on the scheduling of SeismogramSynthesis jobs because, on many of the nodes available on the grid, the amount of memory available per CPU core is only about 1 GB. Thus, if one SeismogramSynthesis job is scheduled on each core, the node could run out of memory. This may explain some of the failures that occurred when the application was run on a different cluster that was relatively memory-poor. Care must be taken if the application is moved to another cluster in the future to configure the scheduler such that it never schedules more SeismogramSythesis jobs on a node than the node can support.

4.3. Broadband

Fig. 5 shows a small Broadband workflow, which integrates earthquake motion simulation codes and calculations. The workflow can be divided into four stages based on the types of computations performed in each stage.

The first stage consists of rupture generation jobs ucsb_createSRF and urs_createSRF. As mentioned earlier, one of the goals of the Broadband platform is to consider different codes that perform similar calculations (based on the underlying models) and compare and verify these calculations. In the example workflow shown in Fig. 5, both types of rupture generation jobs take as input the same simple earthquake description (location and magnitude) and generate time series data that represent skip time histories for the earthquake at the same recording station. The second stage includes the urs_lp_seisgen jobs that calculate deterministic low frequency (up to 1 Hz) seismograms based on the time series data generated in the previous stage. The third stage includes urs_hf_seisgen and sdsu_hf_seisgen, which add stochastic high frequency seismograms to the low frequency seismograms generated by urs_lp_seisgen. The ucsb_seisgen job may be considered to be part of both the second and third stages, as it calculates and merges both low frequency and high frequency seismograms. The fourth stage consists of rspectra jobs that extract parameters from the seismograms that are of interest to earthquake engineers. These parameters may include peak acceleration, peak ground velocity and peak spectral acceleration.

Table 5 shows profile data for each type of job in the Broadband workflow. These statistics were collected from a workflow that simulated the ground motion at 8 stations for 6 earthquakes and a single velocity model. The jobs from the second and third stages (urs_lp_seisgen, sdsu_hf_seisgen, ucsb_seisgen) that generate low and high frequency seismograms had the largest runtimes, with the runtime of the sdsu_hf_seisgen jobs being the largest overall. These jobs operate on relatively simple earthquake descriptions. The sdsu_hf_seisgen job is similar to the SeismogramSynthesis job in CyberShake. More than half of the CPU time of Broadband (∼55%) is used by sdsu_hf_seisgen, making it a good target for optimization. The sdsu_hf_seisgen jobs also generate the most output I/O, on average 1.8 GB per job. In addition, each sdsu_hf_seisgen job uses 942 MB of memory, which makes it difficult to schedule on nodes with only 1 GB per core (keeping in mind that the OS needs some memory for system processes and the kernel). The ucsb_seisgen job consumes the most input (11.2 GB per job).

The Broadband workflow is a good example of how workflow traces and profiles can be used to diagnose unusual and erroneous behavior. For example, while Table 5 shows that the I/O read for ucsb_seisgen is 11.2 GB, it is known that the input file for ucsb_seisgen is only ∼3 GB; therefore at least some of the data are being read multiple times. This suggests that ucsb_seisgen could be optimized to reduce this duplication, which might improve performance. The file system cache may be able to absorb many of these reads. However, if the file system being used does not support client-side caching, as is the case in some parallel file systems, these duplicate reads may greatly reduce performance. Another interesting thing to notice about ucsb_seisgen is its

Table 5
Example of a Broadband execution profile.

Job	Count	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.
ucsb_createSRF	48	0.57	0.13	4.55	0.05	7.45	0.17	6.97	0.01	61.93	0.11
urs_createSRF	48	0.43	0.09	0.00	0.00	1.96	0.01	7.45	0.01	62.98	0.14
urs_lp_seisgen	96	84.92	6.69	43.65	18.30	0.63	0.00	10.13	0.07	91.73	0.06
urs_hf_seisgen	96	3.06	0.63	5.54	0.06	3.31	0.00	12.71	0.09	70.84	0.07
sdsu_hf_seisgen	96	196.40	11.14	938.28	0.01	1855.82	0.01	942.30	0.01	87.72	0.08
ucsb_seisgen	98	68.32	12.43	11 271.46	1292.00	5.11	0.15	53.68	4.90	90.57	0.08
rspectra	288	0.34	0.21	0.81	0.00	0.46	0.00	7.92	0.01	16.20	0.06

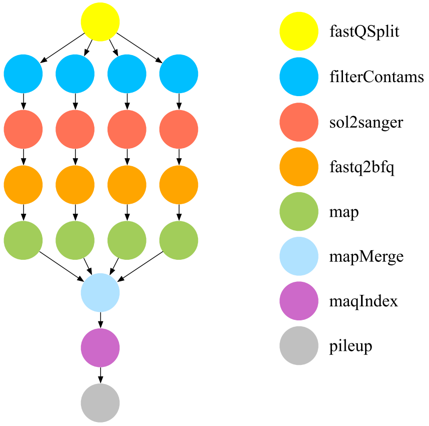

Fig. 6. Epigenomics workflow. The number of splits generated by the first job in the workflow depends on the data partitioning factor used on the input data.

memory usage. The peak memory actually used was 53 MB, but the peak memory allocated was 1847 MB (not shown in the table but reported by pprof). This large gap suggests that there is either a bug in ucsb_seisgen, or that the author intended to cache more of the input data in memory, but the feature was only partially implemented.

4.4. Epigenomics

In Fig. 6, we show an example of an Epigenomics workflow, which maps the epigenetic state of human cells. The Epigenomics workflow is a highly pipelined application with multiple pipelines operating on independent chunks of data in parallel. The input to the workflow is DNA sequence data obtained for multiple "lanes" from the genetic analysis process. The information from each lane is split into multiple chunks by the fastQSplit jobs. The number of splits generated depends on the partitioning factor used on the input data. The filterContams jobs then filter out noisy and contaminated data from each of the chunks. The data in each chunk are then converted to a format understood by the Maq DNA sequence mapping software by the sol2sanger utility. For faster processing and reduced diskspace usage, the data are then converted to the binary fastQ format by fastq2bfq. Next, the remaining sequences are aligned with the reference genome by the map utility. The results of individual map processes are combined using one or more stages of mapMerge jobs. After merging the maqIndex utility operates on the merged alignment file and retrieves reads about a specific region (e.g. chromosome 21). Finally, the pileup utility reformats the data so that it can be displayed by a GUI.

The map jobs that align sequences with the reference genome are the most computationally intensive, followed by the pileup and maqIndex jobs that work on the entire aligned output. The performance of other jobs in the pipeline mainly depends on the amount of data in each of the individual chunks.

Table 6 lists profiling data collected during executions of a sample workflow that was used to align ∼13 million sequences. For most Epigenome jobs, the CPU utilization is high, so we classify the workflow as CPU-bound. The few low utilization jobs are simple data conversion jobs; fastQSplit just splits an input file into multiple output files, and sol2sanger just converts its input file into another format. The pileup job has a mean utilization of 153%; it is possible for a job to have a CPU utilization of more than 100% if it makes use of multiple cores. In this case, pileup is a shell script that pipes the output of Maq to the AWK data extraction utility. Since Maq and AWK are independent processes, they can be scheduled on different cores concurrently, provided that one is not always blocked waiting for the output of another, which appears to be the case here. In addition, knowing that both processes run at the same time is useful from a workflow scheduling perspective because it suggests that perhaps pileup should be treated as a parallel job requiring two cores. However, for this application there is only one pileup job per workflow, so special treatment is not likely to have a significant impact on overall workflow performance. The map jobs represent the vast majority of the runtime of the Epigenome workflow (∼96%), which suggests that any performance optimizations would be best directed towards improving the mapping code.

4.5. LIGO Inspiral Analysis Workflow

The LIGO Inspiral Analysis workflow analyzes data from the coalescing of compact binary systems such as binary neutron stars and black holes. This workflow is very complex and is composed of several sub-workflows. A simplified representation of the workflow is shown in Fig. 7 with sub-workflows outlined by boxes. The actual workflow that was profiled contains 25 sub-workflows with more than 2000 total jobs. Other instances of the workflow contain up to 100 sub-workflows and 200,000 jobs.

The TmpltBank jobs, which identify the continuous family of waveforms that belong to the parameter space for each block of the data, can all be executed in parallel once the input data from the LIGO detectors have been split into multiple blocks (each of 2048 s [3]). The output of a TmplBank job is a bank of waveform parameters that are used by the matched filtering code in an Inspiral job. The triggers produced by multiple Inspiral jobs are tested for consistency by inspiral coincidence analysis jobs, which are denoted by Thinca in the example. Since these jobs operate on data obtained from multiple jobs, they can be regarded as data aggregation jobs. The outputs of the Thinca jobs are inputs to the TrigBank jobs that generate template banks out of the triggers. These template banks are then used by the second set of Inspiral jobs, followed by another set of Thinca jobs. Jobs in the LIGO workflow are described further by Capano [48].

The profiling data for the LIGO Inspiral workflow is shown in Table 7. Inspiral jobs that execute the matched filter to generate triggers are the most computationally intensive jobs in the workflow; the 358 instances of the Inspiral job collectively consume ∼92% of the total runtime and ∼92% of the I/O (197 GB)

Table 6
Example of an epigenomics execution profile.

Job	Count	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.
fast2bfq	128	1.40	0.24	10.09	1.73	2.22	0.40	4.05	0.01	88.42	0.10
pileup	1	55.95	0.00	151.82	0.00	83.95	0.00	148.26	0.00	153.43	0.00
mapMerge	8	11.01	12.18	27.68	32.49	26.71	32.89	5.00	0.39	95.08	0.06
map	128	201.89	21.91	138.76	0.80	0.90	0.22	196.04	5.50	96.69	0.01
sol2sanger	128	0.48	0.14	13.15	2.25	10.09	1.73	3.79	0.00	65.17	0.12
filterContams	128	2.47	0.43	13.25	2.26	13.25	2.26	2.97	0.06	88.54	0.09
mapIndex	1	43.57	0.00	214.10	0.00	107.53	0.00	6.17	0.00	99.50	0.00
fastQSplit	7	34.32	8.94	242.29	82.60	242.29	82.60	2.80	0.00	22.41	0.03

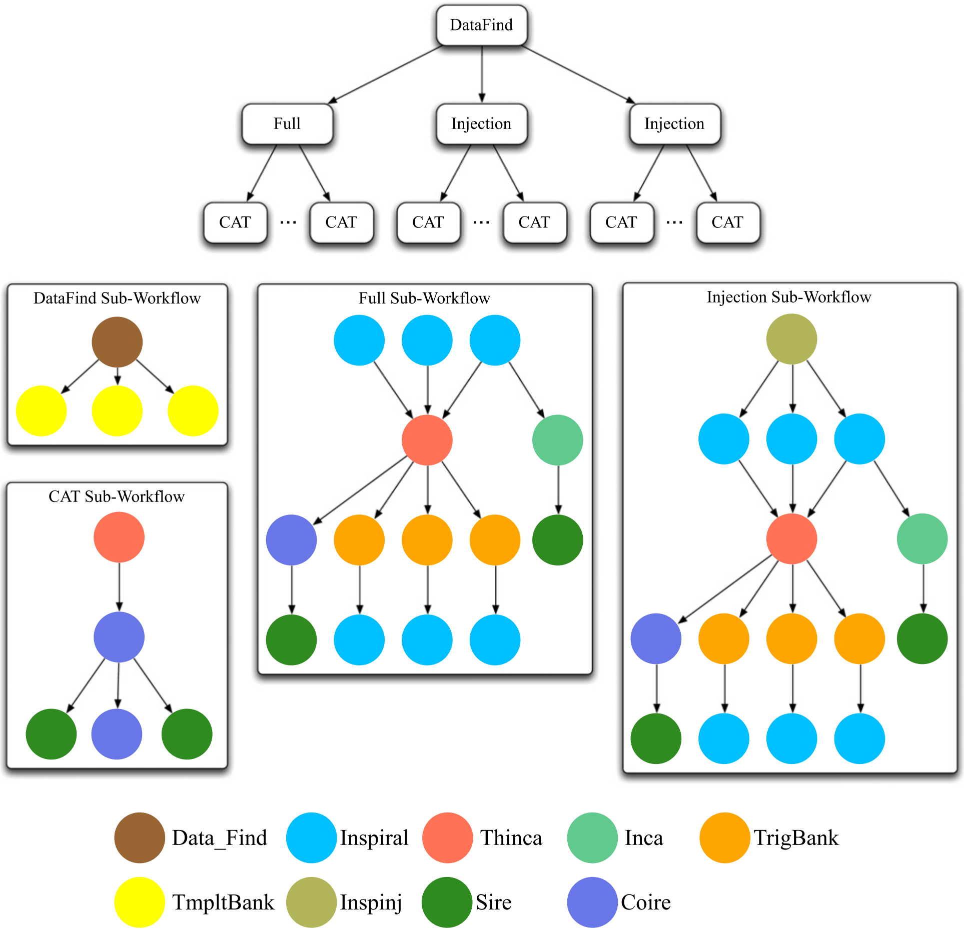

Fig. 7. LIGO Inspiral Analysis workflow. The LIGO workflow is composed in a hierarchical fashion with a single top-level workflow containing several sub-workflows. The sub-workflows in this figure are outlined by boxes.

Table 7
Example of a LIGO execution profile.

Job	Count	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.
TmpltBank	29	497.23	45.78	552.66	6.04	0.04	0.00	404.54	0.02	98.94	0.00
Inspiral	358	472.89	453.18	553.16	5.44	0.05	0.07	533.17	116.28	89.96	0.11
Thinca	290	0.64	1.06	0.53	0.92	0.00	0.01	2.63	0.83	43.90	0.28
Inca	20	0.35	0.27	0.26	0.18	0.13	0.09	2.30	0.35	37.93	0.20
Data_Find	6	0.99	0.38	0.03	0.00	0.01	0.01	10.06	0.01	55.55	0.05
Inspinj	4	0.30	0.28	0.00	0.00	0.08	0.00	1.99	0.01	8.32	0.08
TrigBank	200	0.13	0.14	0.03	0.05	0.00	0.00	2.04	0.14	17.44	0.14
Sire	748	0.25	0.21	0.17	0.34	0.02	0.10	1.93	0.15	14.15	0.18
Coire	386	0.21	0.16	0.07	0.08	0.02	0.03	1.91	0.06	8.00	0.07

and have relatively high CPU utilization (∼90%). The 29 TmpltBank jobs are also computationally intensive (average runtime 500 s and average CPU utilization ∼99%) and read significant input data (16 GB). The remaining jobs in the workflow have short runtimes, low I/O consumption and low CPU utilization. An interesting aspect of the LIGO Inspiral workflow's I/O usage is that it reads a large

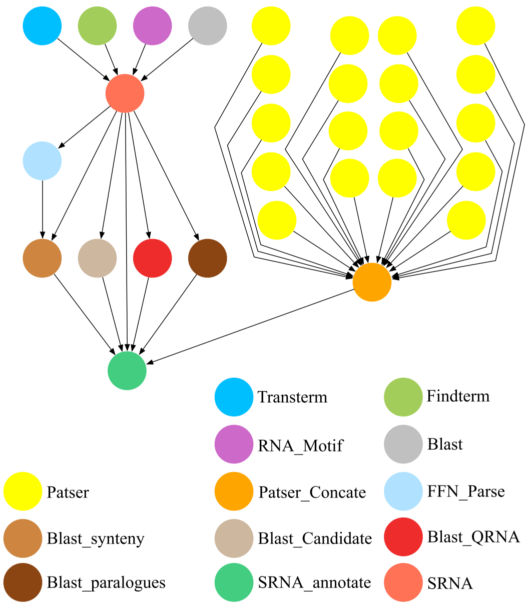

Fig. 8. SIPHT bioinformatics workflow.

amount of data (213 GB), but writes very little (50 MB); the other workflows we characterized generate significant output data.

4.6. SIPHT

A small SIPHT workflow that searches for small untranslated RNAs (sRNAs) is shown in Fig. 8. All SIPHT workflows have almost identical structure, and large workflows can be composed of smaller, independent workflows. The only difference between two workflow instances is the number of Patser jobs, which scan sequences with position-specific scoring matrices that return matching positions; the number of Patser jobs depends on inputs describing transcription factor binding sites (TFBSs). The results of these Patser jobs are concatenated by the Patser_Concate job. There are several BLAST jobs in the workflow that compare different combinations of sequences. The Blast job and the Blast_QRNA job operate on all possible partner inter-genetic regions (IGRs) from other suitable replicons. Even though it is not apparent in Fig. 8, these BLAST jobs operate on hundreds of data files. There are three jobs in the workflow that search for transcription terminators—FindTerm, RNAMotif and Transterm. The sRNA prediction is performed by the SRNA job that operates on the outputs of the above jobs as well as the output of the Blast job. The output of this job is used by the other BLAST jobs. The SRNA Annotate job annotates candidate sRNA loci that were found, for multiple features such as its conservation in other bacterial strains, its association with putative TFBSs, and its homology to other previously identified sRNAs [42]. Of all the jobs in the workflow, the BLAST jobs that compare sequences between several replicon pairs are the most computationally intensive. The next most computationally intensive is the sRNA prediction job, followed by the jobs that identify transcription terminators.

In Table 8, we show the execution profile statistics collected from an execution of the SIPHT workflow. Like the other workflows, SIPHT has a single job type (Blast) that accounts for most of its runtime (∼74%). Finderm and Blast_QRNA also contribute a significant amount to the runtime. Also, similar to Epigenome, SIPHT is primarily a CPU-bound workflow. Most of the jobs have high CPU utilization and relatively low I/O. Only Patser_concate, which simply concatenates the output of the Patser jobs, has low CPU utilization. The Blast and Blast_QRNA jobs read the most input data (∼800 MB each) and write the most output data (∼565 MB each). The Findterm job also writes significant output data (379 MB).

5. Related work

Berry et al. [49] characterized the workloads of several scientific applications; their benchmarks were mainly used to measure hardware performance metrics on supercomputers.

In the area of characterizing workloads in distributed and Grid environments, the Parallel Workload Archive [50] and the Grid Workload Archive [51] provide workloads from parallel and Grid execution environments that can be used in simulations; these workloads focus on the performance and utilization of computational resources. Iosup et al. [52] describe the system-wide, virtual organization-wide, and per user characteristics of traces obtained from four Grids; these analyses provide insight into how Grid environments are used and allow users to model such environments.

Van der Aalst and Ter Hofstede started The Workflow Patterns Initiative to identify basic workflow components [12]. They categorize perspectives such as control flow, data, resource and exception handling that are supported by workflow description and business process modeling languages. They maintain descriptions of common patterns for each perspective, such as sequence, parallel split, synchronization, etc. for control flow.

There have been relatively few efforts to collect and publish traces and performance statistics for real scientific workflows. We recently published traces for a few workflows executed using Pegasus [53] and synthetic workflows based on statistics from real applications for use in simulations [54]. Similarly, Ramakrishnan and Gannon [55] have provided workflow structures and runtime and data statistics for many real workflow applications.

A few systems have been developed to collect profiling data for scientific workflows. Many workflow systems have been integrated with performance monitoring and analysis tools [56–58]. These systems typically collect only coarse-grained information, such as task runtime and data size. Similarly, Kickstart [59] is a tool that is used to monitor task execution in scientific workflows. It captures task information such as runtime, exit status, and stdout/stderr, as well as execution host information, such as hostname, CPU type, and load. Kickstart does not, however, collect fine-grained profiling data. One system that does collect fine-grained profiles is the ParaTrac system [60]. Paratrac uses a FUSE-based file system to monitor and record the I/O usage of tasks and uses the Linux taskstats interface to record memory, CPU, and runtime data. The main drawbacks of the ParaTrac approach are that FUSE is not supported on many of the computing systems used to run workflows, and it only captures I/O performed through the FUSE mount point, which excludes a potentially large amount of I/O. In comparison, our approach collects fine-grained profiling data using the ptrace() API, which is available on most UNIX systems and captures all the operations performed by the task.

Finally, we describe efforts to create repositories for sharing workflow components for use by communities. De Roure et al. [61] describe the myExperiment virtual research environment [62] that enables scientists to share and execute workflows and discuss different workflow management systems and best practices. Von Laszewski and Kodeboyina [63] describe the framework of a Repository Service for Grid Workflow Components that supports storing and sharing of workflow components defined by the community.

Table 8
Example of a SIPHT execution profile.

Job	Count	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.
Patser	19	0.96	0.08	2.70	0.00	0.00	0.00	4.48	0.00	83.48	0.04
Patser_concate	1	0.03	0.00	0.14	0.00	0.14	0.00	2.29	0.00	18.89	0.00
Transterm	1	32.41	0.00	2.93	0.00	0.00	0.00	16.03	0.00	94.79	0.00
Findterm	1	594.94	0.00	15.14	0.00	379.01	0.00	58.21	0.00	95.20	0.00
RNAMotif	1	25.69	0.00	2.91	0.00	0.04	0.00	4.38	0.00	95.05	0.00
Blast	1	3311.12	0.00	808.17	0.00	565.06	0.00	116.38	0.00	93.87	0.00
SRNA	1	12.44	0.00	47.98	0.00	1.32	0.00	5.65	0.00	93.48	0.00
FFN_parse	1	0.73	0.00	5.03	0.00	2.51	0.00	5.00	0.00	81.09	0.00
Blast_synteny	1	3.37	0.00	1.76	0.00	0.42	0.00	14.18	0.00	61.01	0.00
Blast_candidate	1	0.60	0.00	0.27	0.00	0.06	0.00	13.28	0.00	43.61	0.00
Blast_QRNA	1	440.88	0.00	804.60	0.00	567.01	0.00	115.21	0.00	87.80	0.00
Blast_paralogues	1	0.68	0.00	0.12	0.00	0.03	0.00	13.34	0.00	44.30	0.00
SRNA_annotate	1	0.14	0.00	0.40	0.00	0.03	0.00	6.95	0.00	55.96	0.00

6. Conclusions and future work

We characterized six scientific workflows that span a range of characteristics, from small to large and from compute-intensive to data-intensive, with both moderate and large resource requirements. We described our profiling strategy for workflows, which consists of running both I/O and system call trace utilities and analyzing the resulting traces as well as measuring runtimes without the profiling tools. We used these tools to profile the execution of six scientific workflows running at typical scale in grid environments, characterizing their runtimes, I/O requirements, memory usage and CPU utilization based on the tasks within each workflow.

While these workflows are diverse, we found that each workflow had one job type that accounted for most of the workflow's runtime. We also saw evidence that the same data are being re-read multiple times in several of the measured workflows. Both observations suggest opportunities for optimizing the performance of these workflows. In general, workflow profiles can be used to detect errors and unexpected behavior and to identify the best candidates for optimizing workflow execution.

This work is intended to provide to the research community a detailed overview of the types of scientific analyses that are being run using workflow management systems and their typical resource requirements, with the goal of improving the design and evaluation of algorithms used for resource provisioning and job scheduling in workflow systems. In the future, these workflow profiles could be used to create workflow simulations; to construct realistic synthetic workflows that can be used to evaluate different workflow engines and scheduling algorithms; to debug and optimize workflow performance; and to create a suite of benchmarks for scientific workflows.

References

[1] I. Taylor, E. Deelman, D. Gannon, M. Shields, Workflows in e-Science, Springer, 2006.

[2] G.B. Berriman, et al. Montage: a Grid enabled engine for delivering custom science-grade mosaics on demand, in: SPIE Conference 5487: Astronomical Telescopes, 2004.

[3] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal, K. Kennedy, Task scheduling strategies for workflow-based applications in Grids, in: International Symposium on Cluster Computing and the Grid, CCGrid 2005, 2005, pp. 759–767.

[4] L.C. Canon, E. Jeannot, R. Sakellariou, W. Zheng, Comparative evaluation of the robustness of DAG scheduling heuristics, in: S. Gorlatch, P. Fragopoulou, T. Priol (Eds.), Grid Computing, Achievements and Prospects, Springer, 2008, pp. 73–84.

[5] E. Deelman, G. Singh, M. Livny, B. Berriman, J. Good, The cost of doing science on the cloud: the montage example, in: International Conference for High Performance Computing, Networking, Storage and Analysis, SC 2008, 2008.

[6] P.T. Groth, A distributed algorithm for determining the provenance of data, in: IEEE Fourth International Conference on eScience, eScience'08, 2008, pp. 166–173.

[7] P. Groth, E. Deelman, G. Juve, G. Mehta, B. Berriman, Pipeline-centric provenance model, in: Proceedings of the 4th Workshop on Workflows in Support of Large-Scale Science, WORKS'09, 2009.

[8] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman, J. Good, On the use of cloud computing for scientific workflows, in: IEEE Fourth International Conference on eScience, eScience'08, 2008, pp. 640–645.

[9] R. Huang, H. Casanova, A.A. Chien, Using virtual Grids to simplify application scheduling, in: 20th International Parallel and Distributed Processing Symposium, IPDPS 2006, 2006.

[10] A. Merzky, K. Stamou, S. Jha, D.S. Katz, A fresh perspective on developing and executing DAG-based distributed applications: a case-study of SAGA-based montage, in: Fifth IEEE International Conference on e-Science, e-Science'09, 2009, pp. 231–238.

[11] R. Sakellariou, H. Zhao, E. Deelman, Mapping workflows on Grid resources: experiments with the montage workflow, in: F. Desprez, V. Getov, T. Priol, R. Yahnyapour (Eds.), Grids, P2P and Services Computing, Springer-Verlag New York Inc., 2010, pp. 119–132.

[12] The workflow patterns initiative. http://www.workflowpatterns.com.

[13] J. Yu, R. Buyya, A taxonomy of workflow management systems for Grid computing, Journal of Grid Computing 3 (2005) 171–200.

[14] H.L. Truong, S. Dustdar, T. Fahringer, Performance metrics and ontologies for Grid workflows, Future Generation Computer Systems 23 (2007) 760–772.

[15] S. Callaghan, P. Maechling, P. Small, K. Milner, G. Juve, T.H. Jordan, E. Deelman, G. Mehta, K. Vahi, D. Gunter, Metrics for heterogeneous scientific workflows: a case study of an earthquake science application, International Journal of High Performance Computing Applications 25 (2011) 274–285.

[16] L.M.R. Gadelha Jr., B. Clifford, M. Mattoso, M. Wilde, I. Foster, Provenance management in Swift, Future Generation Computer Systems 27 (2011) 775–780.

[17] P. Missier, C. Goble, Workflows to open provenance graphs, round-trip, Future Generation Computer Systems 27 (2011) 812–819.

[18] Y. Simmhan, R. Barga, Analysis of approaches for supporting the Open Provenance Model: a case study of the Trident workflow workbench, Future Generation Computer Systems 27 (2011) 790–796.

[19] J. Palanca, M. Navarro, A. Garia-Fornes, V. Julian, Deadline prediction scheduling based on benefits, Future Generation Computer Systems 29 (2013).

[20] S. Abrishamia, M. Naghibzadeha, D.H.J. Epemab, Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds, Future Generation Computer Systems 29 (2013) 158–169.

[21] C.C. Hsu, K.C. Huang, F.J. Wang, Online scheduling of workflow applications in Grid environments, Future Generation Computer Systems 27 (2011) 860–870.

[22] M. Wieczorek, A. Hoheisel, R. Prodan, Towards a general model of the multi-criteria workflow scheduling on the Grid, Future Generation Computer Systems 25 (2009) 237–256.

[23] M. Mao, M. Humphrey, Auto-scaling to minimize cost and meet application deadlines in cloud workflows, in: Supercomputing 2011 Conference, SC11, 2011.

[24] S. Ostermann, R. Prodan, T. Fahringer, Dynamic cloud provisioning for scientific Grid workflows, in: 11th ACM/IEEE International Conference on Grid Computing, GRID 2010, 2010, pp. 97–104.

[25] G. Singh, C. Kesselman, E. Deelman, Application-level resource provisioning on the Grid, in: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing, 2006.

[26] E.K. Byun, Y.S. Kee, J.S. Kim, S. Maeng, Cost optimized provisioning of elastic resources for application workflows, Future Generation Computer Systems 27 (2011) 1011–1026.

[27] E. Deelman, et al., Pegasus: mapping large-scale workflows to distributed resources, in: I. Taylor, E. Deelman, D. Gannon, M. Shields (Eds.), Workflows in e-Science, Springer, 2006.

[28] E. Deelman, et al., Pegasus: a framework for mapping complex scientific workflows onto distributed systems, Scientific Programming Journal 13 (2005) 219–237.

[29] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, S. Patil, M. Su, K. Vahi, M. Livny, Pegasus: mapping scientific workflows onto the Grid, in: Across Grids Conference, Nicosia, Cyprus, 2004.
[30] The TeraGrid Project. http://teragrid.org/.
[31] P. Maechling, et al., SCEC CyberShake workflows—automating probabilistic seismic hazard analysis calculations, in: E.D.I. Taylor, D. Gannon, M. Shields (Eds.), Workflows for e-Science, Springer, 2006.
[32] SCEC project, Southern California Earthquake Center. http://www.scec.org/.
[33] S. Callaghan, P. Maechling, E. Deelman, K. Vahi, G. Mehta, G. Juve, K. Milner, R. Graves, E. Field, D. Okaya, T. Jordan, Reducing time-to-solution using distributed high-throughput mega-workflows: experiences from SCEC CyberShake, in: 4th IEEE International Conference on e-Science, e-Science 08, 2008.
[34] E. Deelman, S. Callaghan, E. Field, H. Francoeur, R. Graves, N. Gupta, V. Gupta, T.H. Jordan, C. Kesselman, P. Maechling, J. Me-hringer, G. Mehta, D. Okaya, K. Vahi, L. Zhao, Managing large-scale workflow execution from resource provisioning to provenance tracking: the CyberShake example, in: 2nd IEEE Inter-national Conference on e-Science and Grid Computing, e-Science 06, 2006.
[35] USC Epigenome Center. http://epigenome.usc.edu.
[36] Illumina. http://www.illumina.com.
[37] H. Li, J. Ruan, R. Durbin, Mapping short DNA sequencing reads and calling variants using mapping quality scores, Genome Research 18 (2008) 1851–1858.
[38] Maq: mapping and assembly with qualities. http://www.maq.sourceforge.net.
[39] A. Abramovici, W. Althouse, R. Drever, Y. Gürsel, S. Kawamura, F. Raab, D. Shoemaker, L. Sievers, R. Spero, K. Thorne, R. Vogt, R. Weiss, S. Whitcomb, M. Zucker, LIGO: the laser interferometer gravitational-wave observatory, Science 256 (1992) 325–333. 1992.
[40] LIGO project, LIGO—laser interferometer gravitational wave observatory. http://www.ligo.caltech.edu/.
[41] D.A. Brown, et al., A case study on the use of workflow technologies for scientific analysis: gravitational wave data analysis, in: E.D.I. Taylor, D. Gannon, M. Shields (Eds.), Workflows for e-Science, Springer, 2006.
[42] J. Livny, H. Teonadi, M. Livny, M.K. Waldor, High-throughput, kingdom-wide prediction and annotation of bacterial non-coding RNAs, PLoS One 3 (2008) e3197.
[43] J. Frey, Condor DAGMan: handling inter-job dependencies. http://www.bo.infn.it/calcolo/condor/dagman/.
[44] DAGMan (directed acyclic graph manager). http://www.cs.wisc.edu/condor/dagman/.
[45] Strace system call tracing utility. http://sourceforge.net/projects/strace/.
[46] ptrace(2)–Linux man page. http://linux.die.net/man/2/ptrace.
[47] The linux programmer's guide. www.ibiblio.org/pub/Linux/docs/.../programmers-guide/lpg-0.4.pdf.
[48] C.D. Capano, Searching for gravitational waves from compact binary coalescence using LIGO and virgo data, Ph.D. Dissertation, Syracuse University, 2011.
[49] M.W. Berry, et al., Scientific workload characterization by loop-based analyses, SIGMETRICS Performance Evaluation Review 19 (1992) 17–29.
[50] Parallel workloads archive. http://www.cs.huji.ac.il/labs/parallel/workload/.
[51] Grid workloads archive. http://gwa.ewi.tudelft.nl/pmwiki.
[52] A. Iosup, C. Dumitrescu, D. Epema, H. Li, L. Wolters, How are real grids used? The analysis of four grid traces and its implications, in: 7th IEEE/ACM International Conference on Grid Computing, 2006, pp. 262–269.
[53] Workflow gallery. http://pegasus.isi.edu/workflow_gallery/index.php.
[54] Workflow generator. https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator.
[55] L. Ramakrishnan, D. Gannon, A survey of distributed workflow characteristics and resource requirements, Technical Report TR671, Indiana University, 2008.
[56] H.-L. Truong, T. Fahringer, SCALEA-G: a unified monitoring and performance analysis system for the Grid, Scientific Programming Journal 12 (2004) 225–237.
[57] S. Ostermann, K. Plankensteiner, R. Prodan, T. Fahringer, A. Iosup, Workflow monitoring and analysis tool for ASKALON, in: Grid and Services Evolution, 2009.
[58] Sérgio Manuel Serra da Cruz, Fabricio Nogueira da Silva, Luiz M.R. Gadelha Jr., Maria Claudia Reis Cavalcanti, Maria Luiza M. Campos, Marta Mattoso, A lightweight middleware monitor for distributed scientific workflows, in: IEEE International Symposium on Cluster Computing and the Grid, 2008.
[59] E. Deelman, G. Metha, J.-S. Vöckler, M. Wilde, Y. Zhao, Kickstarting remote applications, in: International Workshop on Grid Computing Environments, 2006.
[60] N. Dun, K. Taura, A. Yonezawa, ParaTrac: a fine-grained profiler for data-intensive workflows, in: 19th ACM International Symposium on High Performance Distributed Computing, HPDC 2010, 2010.
[61] D. De Roure, C. Goble, R. Stevens, Designing the myexperiment virtual research environment for the social sharing of workflows, in: 3rd IEEE International Conference on e-Science and Grid Computing, e-Science 07, 2007.
[62] The myExperiment virtual research environment. http://www.myexperiment.org.
[63] G. von Laszewski, D. Kodeboyina, A repository service for Grid workflow components, in: Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services, 2005.

Gideon Juve is a Computer Scientist at the USC Information Sciences Institute. He earned B.S., M.S., and Ph.D. degrees in Computer Science from the University of Southern California. His research focuses on resource management techniques for scientific workflows in clusters, grids, and clouds.

Ann Chervenak is a Research Associate Professor in the USC Computer Science Department and a Project Leader at the USC Information Sciences Institute. Her research interests include data management for distributed scientific collaborations, intelligent data staging for scientific workflows, policy-based data management, replica catalogs, and medical informatics.

Ewa Deelman is a Research Associate Professor at the USC Computer Science Department and a Project Leader at the USC Information Sciences Institute. Dr. Deelman's 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, Dr. Deelman is leading the Pegasus project, which designs and implements workflow mapping techniques for large-scale applications running in distributed environments. Pegasus is being used today in a number of scientific disciplines, enabling researches to formulate complex computations in a declarative way. Over the years, Dr. Deelman worked with a number of application domains including astronomy, bioinformatics, earthquake science, gravitational-wave physics, and others. As part of these collaborations, new advances in computer science and in the domain sciences were made. For example, the data intensive workflows in LIGO (gravitational-wave physics) motivated new workflow analysis algorithms that minimize workflow data footprint during execution. On the other hand, improvements in the scalability of workflows enabled SCEC scientists (earthquake science) to develop new physics-based seismic hazard maps of Southern California. In 2007, Dr. Deelman edited a book on workflow research: "Workflows in e-Science: Scientific Workflows for Grids", published by Springer. She is also the founder of the annual Workshop on Workflows in Support of Large-Scale Science, which is held in conjunction with the Super Computing conference. In 1997 Dr. Deelman received her Ph.D. in Computer Science from the Rensselaer Polytechnic Institute.

Shishir Bharathi received a B.E. in Computer Science and Engineering from Mysore University, India. He then received an M.S. and a Ph.D. in Computer Science from the University of Southern California. His research focus was on efficient data and information management for the execution of large scale scientific workflows on the Grid. He currently works at Amazon.com.

Gaurang Mehta is a Research Programmer at USC Information Sciences Institute. He is a member of the Collaborative Computing Group, at the Information Sciences Institute and works on the Pegasus Project (http://pegasus.isi.edu) as well as Center for Genetics Studies on Mental Disorders (https://www.nimhgenetics.org). He received a M.S. in Electrical Engineering from University of Southern California and a B.E. in Electronics Engineering from Mumbai University, India. Gaurang currently leads and develops solutions for several BioInformatics Projects. Gaurang has been a member of the IEEE for the last 15 years.

Karan Vahi is a Research Programmer at USC Information Sciences Institute. He is a member of the Advanced Systems Division at the Information Sciences Institute and works on the Pegasus Project. He received a M.S in Computer Science from University of Southern California and a B.E in Computer Engineering from Thapar University, India. Karan has been associated with the Pegasus project since its inception, first as a Graduate Research Assistant and then as a full time programmer after graduating from USC. He is currently the lead developer on it and works closely with the user community to drive its development. He is the co-author on most of Pegasus research publications. Before joining USC, Karan worked briefly at Quark Media House as a programmer. There he worked on a business workflow management product.