Scientific Programming 18 (2010) 77–92
DOI 10.3233/SPR-2010-0300
IOS Press

Experiences with resource provisioning for scientific workflows using Corral

Gideon Juve, Ewa Deelman *, Karan Vahi and Gaurang Mehta
Information Sciences Institute, University of Southern California, Marina Del Rey, CA, USA
E-mails: {gideon, deelman, vahi, gmehta}@isi.edu

Abstract. The development of grid and workflow technologies has enabled complex, loosely coupled scientific applications to be executed on distributed resources. Many of these applications consist of large numbers of short-duration tasks whose runtimes are heavily influenced by delays in the execution environment. Such applications often perform poorly on the grid because of the large scheduling overheads commonly found in grids. In this paper we present a provisioning system based on multi-level scheduling that improves workflow runtime by reducing scheduling overheads. The system reserves resources for the exclusive use of the application, and gives applications control over scheduling policies. We describe our experiences with the system when running a suite of real workflow-based applications including in astronomy, earthquake science, and genomics. Provisioning resources with Corral ahead of the workflow execution, reduced the runtime of the astronomy application by up to 78% (45% on average) and of a genome mapping application by an order of magnitude when compared to traditional methods. We also show how provisioning can benefit applications both on a small local cluster as well as a large-scale campus resource.

Keywords: Scientific workflows, grid computing, distributed computing, high-throughput computing, web services, pilot jobs, glideins

1. Introduction

Workflow systems have been used to manage large-scale, loosely-coupled scientific computations in a wide variety of domains including physics [10], earth science [2,9], and astronomy [18]. These applications often consist of large numbers of compute- or data-intensive tasks with complex control and data flow dependencies. Completing these computations in a reasonable amount of time requires the use of high-performance computing systems such as clusters and grids [25,35], and more recently clouds [8,17].

Although clouds are being investigated as a computing platform for science applications, the majority still run on campus clusters or grids. Most of these high-performance computing systems provide an execution model based on batch scheduling where jobs are held in a queue until they can be matched with resources for execution. These systems provide only best-effort service and are typically configured to maximize resource utilization and not throughput. As a result, they often impose significant scheduling and queuing overheads on application jobs. For applications that consist of a single parallel job, or a few independent jobs, this model works well because the overheads do not add significantly to the total runtime of the application. For workflows and other high-throughput applications with large numbers of tasks these overheads have a detrimental impact on performance because delays are accumulated many times as the application executes.

One solution to this problem is to use *resource provisioning* [16,31], where resources are allocated at the application level instead of being allocated for each job individually. This enables an application to execute multiple jobs on a set of resources while incurring the overheads associated with acquiring those resources only once. This technique improves the runtime of applications with many jobs by minimizing the impact of queue delays.

Because resource provisioning is not directly supported by most grid sites, a number of systems have been developed that rely on *multi-level scheduling* [16, 27] to enable provisioning on the grid. Multi-level scheduling is a technique in which *pilot jobs* are used to run guest resource managers on worker nodes that belong to a host cluster. This technique enables the

*Corresponding author: Ewa Deelman, Information Sciences Institute, University of Southern California, 4676 Admiralty Way, Suite 1001, Marina Del Rey, CA 90292, USA. Tel.: +1 310 448 8408; Fax: +1 310 823 6714; E-mail: deelman@isi.edu.

use of resource provisioning on almost any grid site. In addition, since multi-level scheduling makes use of a user-level scheduler to manage application jobs, the user can define custom scheduling policies that are beneficial for their application.

The contributions of this paper are:

- The design and implementation of a resource provisioning system that automates the creation of personal Condor [20] pools to manage resources provisioned from grid sites. The system supports useful features such as a scriptable command-line interface, synchronous and asynchronous APIs, and the automatic resubmission of provisioning requests.
- A characterization of the system overheads and their measurement on three TeraGrid systems.
- An evaluation of the benefits of the system using three real workflow applications on a grid cluster and a small local cluster.
- A case study showing how the system is used to provision resources for very large-scale workflow applications.

The experiments show that our system can reduce the runtime of an astronomy application by as much as 78%, of an earthquake science application by 40%, and of a genome mapping application by as much as an order of magnitude when compared to traditional methods.

Although our approach focuses on grid-based systems, the concept of multi-level scheduling is also applicable to cloud-based infrastructure. There, native APIs can be used to provision resources and application schedulers can manage the execution of tasks on them.

2. Overview of provisioning methods

The traditional approach to resource access in grid environments is based on a queuing model that provides best-effort quality of service. In this model jobs are queued until they can be matched with appropriate resources for execution. This approach ensures that access to resources is shared equally and fairly among all users of the system, but can result long delays when competition between users forces jobs to wait for resources to become available. For applications with only one job, or with a few jobs that can be submitted in parallel, these delays are encountered only once. For workflow applications with complex job hierarchies and interdependencies the delays are encountered many times.

One way to improve quality of service for workflow application is to use a model for resource allocation based on provisioning. With a provisioning model, resources are allocated for the exclusive use of a single user for a given period of time. This minimizes queuing delays because the user's jobs no longer compete with other jobs for access to resources. Furthermore, in contrast to the queuing model where resource allocation and scheduling occur on a per-job basis, the provisioning model allows resources to be allocated once and used for multiple jobs.

Provisioning is slightly more complex than queuing in that it requires users to make more sophisticated resource allocation decisions. There are two policies that can be used to guide these decisions. In static provisioning the application allocates all resources required for the computation before any jobs are submitted, and releases the resources after all the jobs have finished. This method assumes that the number of resources required is known or can be predicted in advance. In dynamic provisioning resources are allocated by the application at runtime. This allows the pool of available resources to grow and shrink according to the changing needs of the application. Dynamic provisioning does not require advanced knowledge of resource needs, but it does require policies for acquiring and releasing resources.

2.1. Advance reservation

Advance reservation is a resource provisioning mechanism supported by many batch schedulers [15, 22,26]. Users create advance reservations by requesting slots from the batch scheduler that specify the number of resources to reserve and the start and end times of the reservation. During the reservation period the scheduler only runs jobs that belong to the user on the reserved resources.

Although the batch schedulers used by many resource providers have advance reservation features, few providers support the use of reservations. Singh et al. conducted a survey of advance reservation capabilities at several grid sites [31]. They discovered that 50% of the sites surveyed did not support reservations at all, and that most of the sites that did support reservations required administrator assistance to create them. Only

a few sites allowed users to create their own reservations. This lack of support makes using advance reservations time-consuming and cumbersome.

Scheduler-based advance reservations also increase resource usage costs. In many grid environments these costs are measured in service units. Users of advance reservations are typically charged a premium for dedicated access to resources. These premiums can be 20–100% above normal costs [28]. Furthermore, users are often forced to pay for the entire reservation, even if they are not able use it all (e.g., if there is a failure that causes the application to abort, or if the actual runtime of the application is shorter than predicted).

An alternative to scheduler-based advance reservations is the use of probabilistic advance reservations [24]. In this method reservations are made based on statistical estimates of queue times. The estimates allow jobs to be submitted with a high probability of starting some time before the desired reservation begins. This allows "virtual reservations" to be created by adjusting the runtime of the job to cover both the time between the submission of the job and the desired reservation start time, and the duration of the reservation itself.

Unlike scheduler-based reservations, probabilistic reservations do not require special support from resource providers. However, probabilistic reservations are not guaranteed because the actual queue delay may exceed the predicted delay, and the final cost of a probabilistic reservation is difficult to predict because the actual runtime of the reservation job may exceed the desired reservation time.

2.2. Multi-level scheduling

Many of the performance issues encountered by workflow applications on the grid arise because resource providers control both the management of resources and the scheduling of application jobs. In *multi-level scheduling* [16,27] these two functions are separated. Providers retain authority over resources, but users are given control over scheduling. This division is accomplished by creating *personal clusters* [19, 38] to reserve and manage resources. Personal clusters are pools of resources that are temporarily leased from a resource provider and managed by an application-level scheduler.

The process of creating a personal cluster is illustrated in Fig. 1. A user requests resources from a resource provisioner, which submits pilot jobs to a grid site. From the perspective of the site's scheduler, these jobs are indistinguishable from normal computing jobs. However, instead of running an application program, the pilot jobs install and run guest node managers on the site's worker nodes. On startup, the guest managers register themselves with an application-level scheduler that is controlled by the user. The newly registered nodes become part of the user's personal cluster and are matched with application jobs for execution.

One advantage of this approach is that it minimizes the overheads caused by queuing. Any delays caused

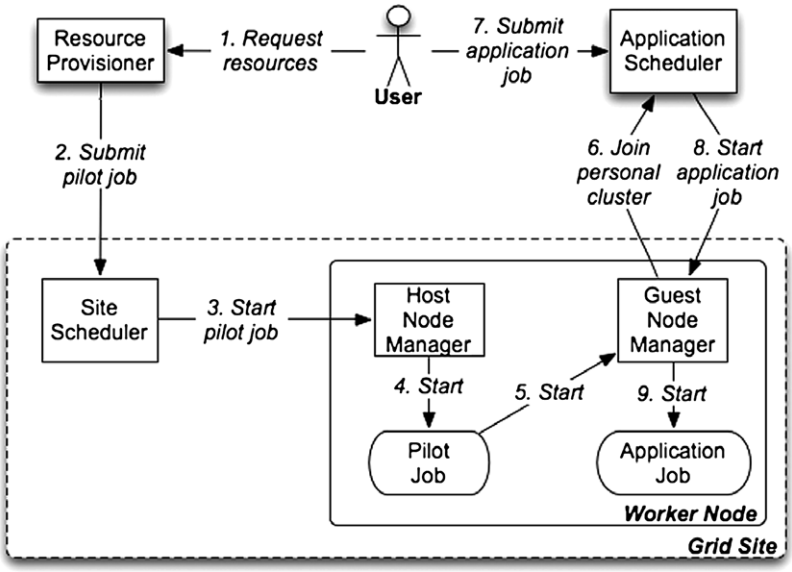

Fig. 1. Multi-level scheduling.

by queuing are encountered only once when the pilot job is submitted. After the pilot job starts, the resources it reserves can be used to execute multiple application jobs. By reusing provisioned resources the total delay experienced by the application can be significantly reduced.

Another advantage of this approach is that it allows the user to define custom policies and configurations that reduce scheduling overheads. Because application jobs do not pass through the site's scheduler they avoid many of the overheads associated with traditional resource access methods such as multiple layers of software involved in job submission and polling operations to check for job status changes. In addition, scheduling policies can be fine-tuned to fit the specific characteristics of the application. For example, workflow jobs with many dependants could be given priority over jobs with fewer dependants in order to generate opportunities for parallelism more quickly. Similarly, jobs at a higher level of the workflow could be given priority over jobs later in the workflow to improve pipelining. Also, using a custom scheduler allows applications to take advantage of the many sophisticated task-scheduling algorithms available [1,3,21]. Together these optimizations can result in significant decreases in workflow runtime. Singh et al. has demonstrated some of the benefits of using application-specific scheduling parameters for workflow execution in [30].

3. Corral

In this section we describe the design and implementation of a resource provisioning system based on multi-level scheduling called Corral. Corral creates personal clusters by provisioning resources from grid sites using pilot jobs to acquire resources. Once these jobs are running on the remote resources, they start up daemons that contact the personal cluster manager and make themselves available to run application tasks.

In our case we use Condor *glideins* [13], where Condor worker daemons are started on remote cluster nodes. Upon startup, the worker daemons join a Condor pool administered by the user (i.e., a personal cluster) where they are used to execute application jobs. The use of this technique has been shown to reduce runtimes for several large-scale workflow applications [30,31].

3.1. Design goals

Based on our analysis of several existing multi-level scheduling systems (see Section 5) and the requirements of workflow applications we developed the following list of goals for our system:

- **Automate environment setup**. Rather than relying on the user to install software on the remote site, our system should automate the setup process as much as possible while allowing the user to control details of the configuration.
- **Minimize overheads**. Several of the existing systems transfer large executables for each provisioning request, introducing overheads that delay the provisioning of resources. Our system should try to minimize these delays by reducing the amount of data transferred for each request and by supporting the allocation of multiple processors in one request.
- **Provide multiple interfaces**. Few of the existing systems support programmatic access to provisioning functions. Our solution should provide a complete programmatic interface that can be used by external software tools in addition to a scriptable, easy-to-use command-line interface.

3.2. Implementation

Corral was developed using a service-oriented architecture. Clients send provisioning requests to a web service, which communicates with grid sites to allocate resources that fulfill the requests. The components of the system and the relationships between them are shown in Fig. 2.

The **Submit Site** has one or more servers that host files and services used in the provisioning process. These servers are owned and controlled by the user or the organization to which they belong.

The **Grid Site** consists of a head node, several worker nodes, and a shared file system that can be accessed by all nodes. The head node hosts a remote job submission interface (e.g., Globus gatekeeper) that accepts batch jobs, and a local resource manager (LRM) that matches jobs with resources.

Corral accepts requests from clients, sets up the execution environment on the grid site, provisions resources using pilot jobs (glideins), and cleans up files and directories created by the system. This service was implemented as a RESTful web service [12].

Condor is used as the personal cluster to process service and application jobs, and to manage glidein

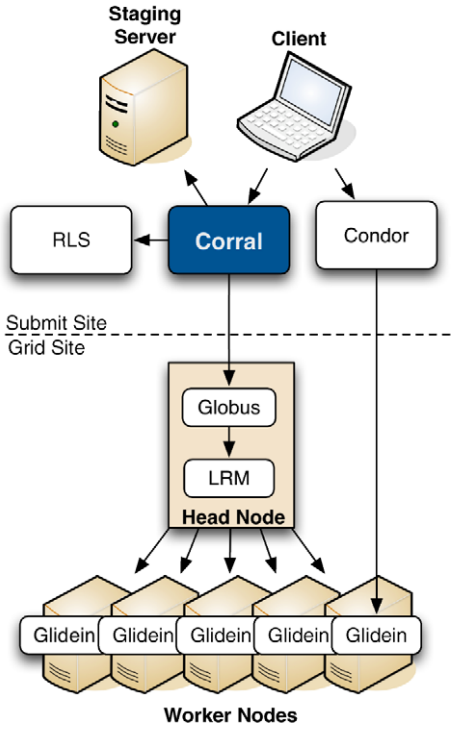

Fig. 2. Corral components.

workers. Corral submits glidein jobs to the grid site using Condor, the glideins contact the Condor central manager to join the user's personal cluster, and application jobs are submitted to the Condor queue where they are matched to glideins for execution.

Staging Servers are file servers used to host bundles of executables and configuration files called *packages*. Each package contains a set of Condor worker node daemons for a different combination of Condor version, system architecture and OS. Any file server that supports the HTTP(S), FTP or GridFTP protocols may be used as a staging server.

A **Replica Location Service** is used to map package names to staging servers, in our case we use **RLS** [4,5].

3.3. Operation

The process used by the service to provision resources is divided into three phases: setup, provisioning, and cleanup. These phases correspond to jobs that are submitted by the service to the grid site.

The **Setup Job** is submitted during the *setup phase* to prepare the site for glideins. The setup job runs an installer which determines the appropriate package to use for the site, looks up the package in RLS to determine which staging servers have it, and downloads the package from the first available staging server. It then creates an installation directory and a working directory on the shared file system, and unpacks the Condor binaries.

The **Glidein Job** is submitted during the *provisioning phase* to allocate worker nodes for the user's personal cluster. Glidein jobs generate a Condor configuration script and launch Condor worker daemons on each allocated node. The daemons are monitored by a special process and killed when the user's request is cancelled or expires.

The **Cleanup Job** is submitted during the *cleanup phase* to remove the working directories used by the glideins. It runs an uninstaller, which removes all log files, configuration files, executables and directories created by the service.

Using this three-step process allows Condor executables to be staged once during the setup phase and reused for multiple requests during the provisioning phase. This precludes the transfer of binaries for each provisioning request and thereby reduces the provisioning overhead of the system.

3.4. Features

Interfaces: Users can interact with the system using a simple command-line interface. In addition to providing functionality for interactive provisioning requests, the command-line interface also supports scripting by providing outputs that are easy to parse and operations that block until resources have been allocated. This allows the command-line interface to be used in shell scripts and workflows to automate provisioning. This capability could be used, for example, to create a meta-workflow that automates the planning and execution of other workflows as shown in Fig. 3. In addition, the service uses a simple RESTful interface that can be invoked directly, and there is a simple client API that can be used in Java applications.

Asynchronous Notifications: The service supports an asynchronous interface that can be used by clients to receive automatic notifications when the state of a request changes.

Automatic Resubmission: Many resource providers limit the maximum amount of time that can be requested for an individual job. This means that pilot jobs used to provision resources can only run for a lim-

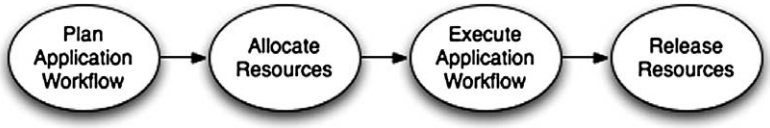

Fig. 3. Example meta-workflow containing resource provisioning jobs.

ited amount of time before they expire. Often, however, users would like to provision resources for a longer time to accommodate long-running applications. This can be accomplished by resubmitting provisioning requests as they expire. Corral supports this capability by automatically submitting new pilot jobs to replace old jobs that have terminated. When creating a new pilots the user can specify that the request should be resubmitted indefinitely, until a specific date and time, or a fixed number of times. When resubmitting, if the last request failed, or if the user's credential has expired, the request will not be resubmitted. Unlike other techniques, such as advance reservations, this approach is not unfair because resubmitted pilot jobs are treated just like any other job. They are not given special priority and must wait in the remote queue alongside other users' jobs.

Firewall Negotiation: Multi-level scheduling systems function well when worker nodes have public IP addresses and are free to communicate with clients outside their network. However, many resource providers conserve IP addresses by using private networks and isolate their worker nodes behind firewalls for security. This prevents application-specific schedulers outside the resource provider's network from communicating directly with worker nodes. Solutions to this problem include Generic Connection Brokering (GCB) [14] and Condor Connection Brokering (CCB) [6]. These techniques make use of a broker that is accessible to both the worker nodes and the application-specific scheduler to facilitate connections between them. The broker allows the application-specific scheduler and the worker nodes to communicate without requiring any direct connections into the private network. Corral supports both GCB and CCB.

4. Evaluation

We performed the evaluation of our approach in four ways:

(1) We measured the overhead of starting up pilot jobs to provision resources, breaking into the phases of setting the software at the site (automated), starting the jobs, and performing cleanup of the site.
(2) We measured the overhead of an application job startup with a pilot job in place.
(3) We measured an end-to-end performance of three real-world workflow-based application executing on Corral-provisioned resources. We compared that runtime with a lower-bound calculated using a well-known DAG scheduling algorithm.
(4) We described our experience running a number of large-scale earthquake science workflows on the TeraGrid in a production mode.

4.1. Resource provisioning overhead

In order to quantify the overhead of using Corral to provision resources we measured the amount of time required to complete the phases of the resource provisioning process. These phases correspond to the lifecycle events of the setup, glidein and cleanup jobs used by Corral as shown in Fig. 4. The provisioning phase is composed of two sub-phases, allocation and runtime, that correspond to the scheduling/queuing delay and the execution time of the glidein job.

We measured the setup time, allocation time and cleanup time for three typical TeraGrid sites (averages shown in Table 1). All allocation measurements were taken when the sites had sufficient free resources in order to minimize the impact of queuing delays. As such these figures represent lower bounds on the allocation time.

The overheads for setup and cleanup were approximately 30 and 15 s, respectively. The uniformity of the results is a result of using the Globus fork jobmanager to run setup and cleanup jobs on all sites, which imposes a small, constant overhead. In comparison, the glidein jobs were submitted the batch jobmanager (i.e., jobmanager-pbs), which is reflected in the variation in allocation time between the sites. This variation is due to scheduling overheads, which depend on site policies and configuration.

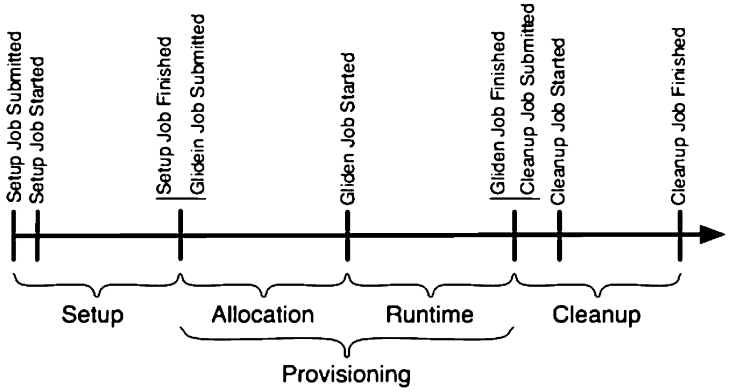

Fig. 4. Resource provisioning phases.

Table 1
Average provisioning overheads (in seconds)

Site	Setup time	Allocation time	Cleanup time
NCSA Mercury	29.5	52.5	15.0
NCSA Abe	28.4	35.3	15.7
SDSC IA-64	28.4	97.0	15.2

Table 2
Average no-op job runtime (in seconds)

Site	GT2	GT4	Glidein
NCSA Mercury	61.1	237.9	2.2
NCSA Abe	35.8	220.7	1.6
SDSC IA-64	263.3	N/A[1]	2.0

Note: [1]The GT4 service was not available at this site.

4.2. Job execution delays

In order to determine the benefits of running application jobs using glideins we measured the amount of time required to run a no-op job using Globus version 2, Globus version 4 and Condor glideins. Please note that the time to set up the glide-in is not included in the Glidein number. Rather this time represent the runtime of a no-op job once the glide-in is acquired. The average runtimes for three TeraGrid sites are shown in Table 2.

On all three sites, the runtime of the jobs using glideins ($\sim$2 s) was significantly shorter than the runtime using Globus ($\sim$35–260 s). This improvement is attributed to two factors: reduced software overhead, and reduced scheduling delay. The reduction in software overhead is a result of Condor requiring fewer software layers to dispatch jobs than Globus. The reduction in scheduling delay results from the ability to configure Condor to immediately execute jobs if there are available resources. In comparison, Globus is limited by the scheduling policies of each site's LRM, which are typically configured to schedule jobs periodically on intervals of up to several minutes.

These measurements clearly show the benefit of a pilot-job approach on Grid-based systems, especially when the overheads shown are incurred by every job in the application workflow.

4.3. Workflow runtime

In the following sections we quantify the benefits of Corral using three real-world workflow applications, running on three different types of execution environments (small and large clusters). We measured the makespan of the workflows (not including the provisioning step, which adds $\sim$100 s to the overall workflow as shown in Table 1). We picked applications from three different disciplines: earthquake science, astronomy and epigenomics. The tasks within these applications can be characterized as memory-intensive, I/O-intensive and CPU-intensive, respectively.

For large workflows (astronomy and epigenomics) we also evaluated the impact of task clustering on overall workflow performance with and without resource provisioning ahead of the execution.

4.3.1. Earthquake science workflow

The first type of application we experimented with was an earthquake science application, Broadband, developed by the Southern California Earthquake Center (SCEC) [33]. The objective of Broadband is to integrate a collection of motion simulation codes and calculations to produce research results of value to earthquake engineers. Broadband workflows combine these codes to simulate the impact of various earthquake scenarios on several recording stations. Researchers use the Broadband platform to combine low frequency

(less than 1.0 Hz) deterministic seismograms with high frequency (∼10 Hz) stochastic seismograms and calculate various ground motion intensity measures (spectral acceleration, peak ground acceleration and peak ground velocity) for their building design procedures. We ran an experiment to compare the performance of Broadband using the provisioning approach to that using the traditional approach. The workflow used in our experiment simulates 6 earthquake scenarios for 8 different recording stations and contains 768 tasks. A smaller example is shown in Fig. 5 to illustrate the structure of a Broadband workflow.

We ran the 768-task workflow on WIND, a small cluster at ISI. WIND nodes have two, 2.13 GHz dual core Intel Xeon CPUs (4 cores total) and 4 GB of memory. We used 4 nodes (16 cores) for the computations. The local scheduler on WIND is Condor 7.1.3. For experiments using the traditional approach jobs were submitted using Globus GT2 GRAM. For experiments using multi-level scheduling we used Corral to provision a fixed number of processors and submitted workflow jobs to a dedicated glidein pool. The Pegasus Workflow Management System [11] was used to plan and execute all experiments.

Figure 6 shows the results of running Broadband with and without provisioning. Using provisioning (Corral), the application runs 40% faster than with the traditional approach where jobs are submitted directly to the job manager on the resource (Globus). For comparison, we also computed a lower bound on the runtime of the workflow using a modified version of the HEFT scheduling heuristic [36]. Our version of HEFT assumes no scheduling or communication overheads and uniform resources. Since these overheads are not included, we claim that the HEFT runtime is a reasonable lower bound on the runtime that could be achieved in a real execution environment where the overheads are present. We can see that the workflow runtime with Corral is very close to the lower bound. However, it is still slower because provisioning cannot eliminate all the overheads, such as the waiting of a workflow task in a queue in the workflow management system, and the delay in sending the task to the computational resource (in this case over an Local Area Network).

4.3.2. Astronomy workflow

The second experiment involved running an astronomy application, Montage [18], using both the traditional grid approach and multi-level scheduling provided by Corral. The input to Montage is the region of the sky for which a mosaic is desired, the size of the mosaic in terms of square degrees, and other parameters such as the image archive to be used, etc. The input images are first re-projected to the coordinate space

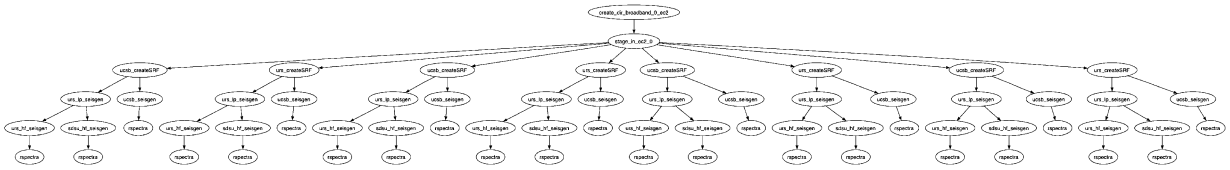

Fig. 5. A view of a small Broadband workflow containing 66 tasks.

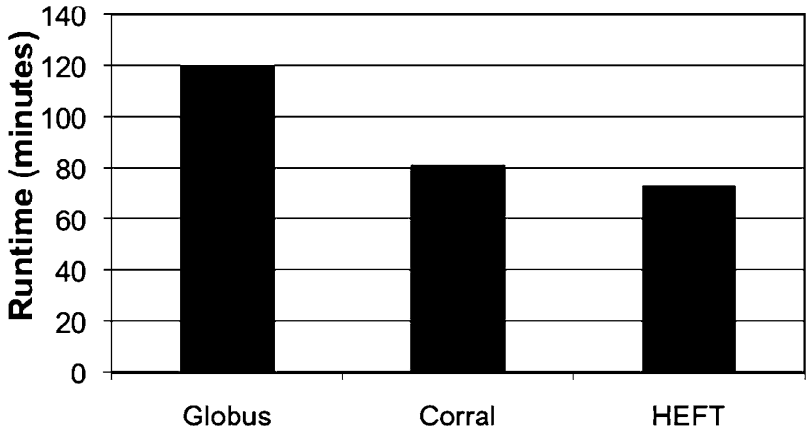

Fig. 6. Execution Time (in minutes) of Broadband on a Local Cluster with and without provisioning. HEFT shows the lower-bound on the execution.

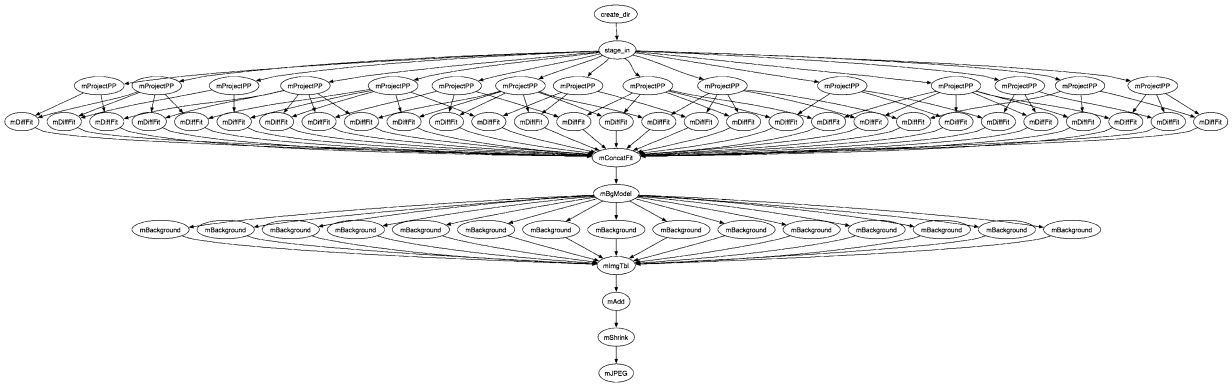

Fig. 7. Small Montage workflow with 67 tasks.

of the output mosaic, the re-projected images are then background rectified and finally co-added to create the output mosaic. Montage is a data-intensive application. The input images, the intermediate files produced during the execution of the workflow, and the output mosaic are of considerable size and require significant storage resources. The tasks, on the other hand, have short runtimes of at most a few minutes.

The size of a workflow (number of tasks) is based on the area of the sky in square degrees covered by the mosaic. We used 2 different sized workflows for our evaluation: a 1-degree workflow with 206 tasks and a 6-degree workflow with 6062 tasks. Figure 7 shows a smaller, 0.5-degree Montage workflow to give the reader an idea about the shape of the larger workflows.

We used the medium-sized Skynet cluster at ISI for all Montage experiments. The nodes on Skynet have 800 MHz Pentium III processors and 1 GB of memory. To isolate the effects of scheduling overhead, which can be prominent in large-scale workflows with short-duration tasks, we performed experiments using both unclustered and clustered versions of the workflows. For the clustered configurations we grouped tasks from each level of the workflow into N jobs where N equals the number of available processors. (Workflow tasks are at the same level if their distance from their respective parent is the same.) Workflows that are automatically clustered in this manner produce the minimum scheduling overhead achievable without reducing workflow parallelism [32].

The results of these experiments are shown in Fig. 8. For unclustered experiments, the runtime of the workflows using Corral was 45% less on average than the runtime using Globus (up to 78% in the best case). The clustered experiments showed a more modest improvement of 11% on average (23% best case). This was primarily due to a decrease in scheduling overheads for the clustered experiments that result from having fewer jobs to schedule. It is also interesting to note the difference between the fine- and coarse-grained workflows. For the clustered experiments, we generated the same number of clusters for all workflow sizes. Because the larger workflows have more total tasks there were more tasks per cluster and the resulting runtimes were larger. This difference in cluster granularity had an impact on the relative scheduling overheads. By comparing with the HEFT runtimes we can see that, for the fine-grained workflows (1-degree) the scheduling overheads dominate the execution, and for the coarse-grained workflows (6-degree) we can achieve performance close to optimal. In addition, the fine-grained workflows have enough scheduling overhead that we do not see much benefit from increasing the number of processors.

4.3.3. Epigenomic workflow

To illustrate the benefits of using Corral for larger workflows and sites with restrictive policies we performed a set of experiments using an Epigenome mapping application [6]. The application consists of 2057 tasks that reformat, filter, map and merge DNA sequences. The majority (>90%) of the runtime of this application is consumed by 512 tasks that require approximately 2.5 h to run each. Figure 9 shows a picture of a much smaller Epigenomic workflow for illustration.

In this case, the application scales well to large numbers of cores and thus we conducted the following experiments on the 10 GB cluster at USC's High-Performance Computing Center (HPCC). The nodes we used had 2.3 GHz AMD Opteron processors and 16 GB memory. HPCC's cluster has two scheduling policies that affect the runtime of the epigenome application: *max_user_run*, and *resources_max.walltime*. The max_user_run policy prevents any single user

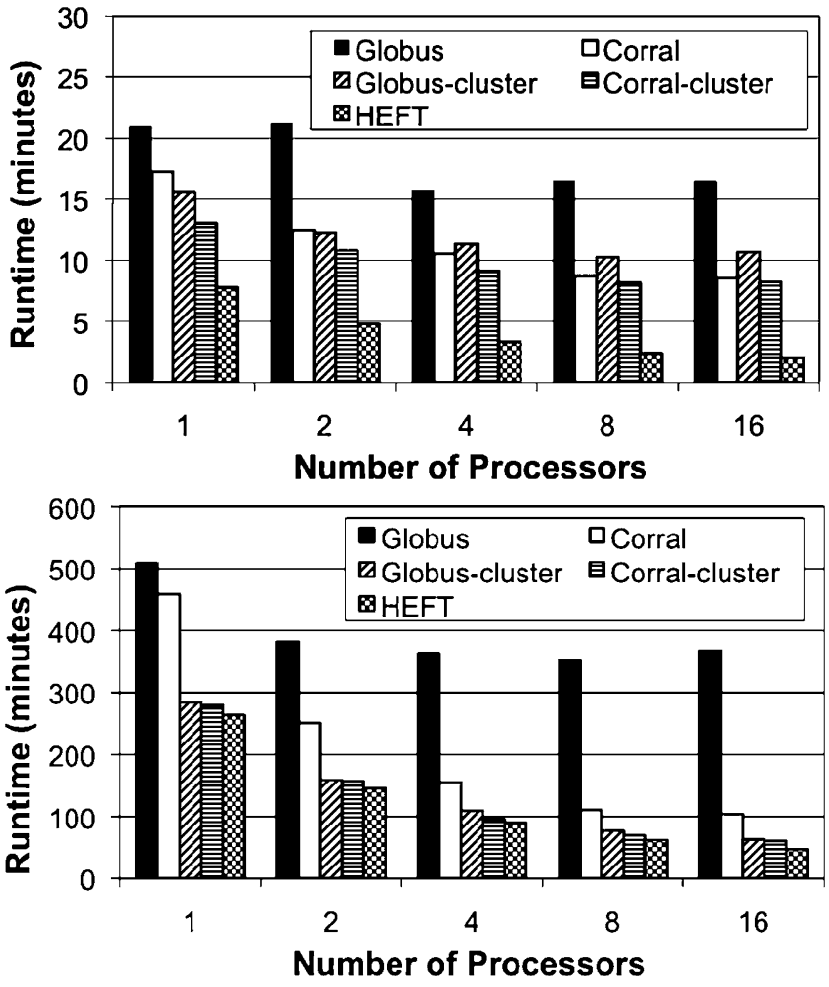

Fig. 8. Comparison of Montage runtimes for a 1-degree workflow (top) and a 6-degree workflow (bottom).

from running more than 30 jobs concurrently. When using Globus to submit jobs, this policy prevents workflows with serial tasks from using more than 30 processors at a time. The *resources_max.walltime* policy prevents any single job from running for more than 24 h. One result of this policy is that workflows with long-running jobs cannot be clustered to match the allowable resources. For the genomic workflow, clustering tasks into 30 jobs per level (to match max_user_run) would result in individual job runtimes of ∼45 h. Consequently, the maximum clustering possible is 60 jobs per level, which results in runtimes of ∼22.5 h. These clustered jobs must be executed in two separate batches of 30 jobs each.

Figure 10 compares the runtime of the genomic workflow when limited to 30 processors. The runtime using Corral was 32% less than Globus in the unclustered case, and 10% less in the clustered case. Using Corral the clustered workflow actually took longer than the unclustered workflow. This is a result of the interaction between provisioning and load imbalance. Due to the limitation on the runtime of the jobs and to make a closer comparison with the plain Globus solution, the resources are provisioned in two 24 h blocks, with the second block being requested when all the jobs scheduled for the first block have completed. However not all the jobs take the same time to complete and the jobs in the second block must wait for the slowest jobs in the first block before they can proceed. The problem occurs for both clustered and unclustered workflows, however the shorter jobs in the unclustered workflow compensate for the imbalance and make the gap less severe. This gap could be eliminated completely by using a more sophisticated provisioning scheme (e.g., by overlapping the blocks or using dynamic provisioning). However, since the over-

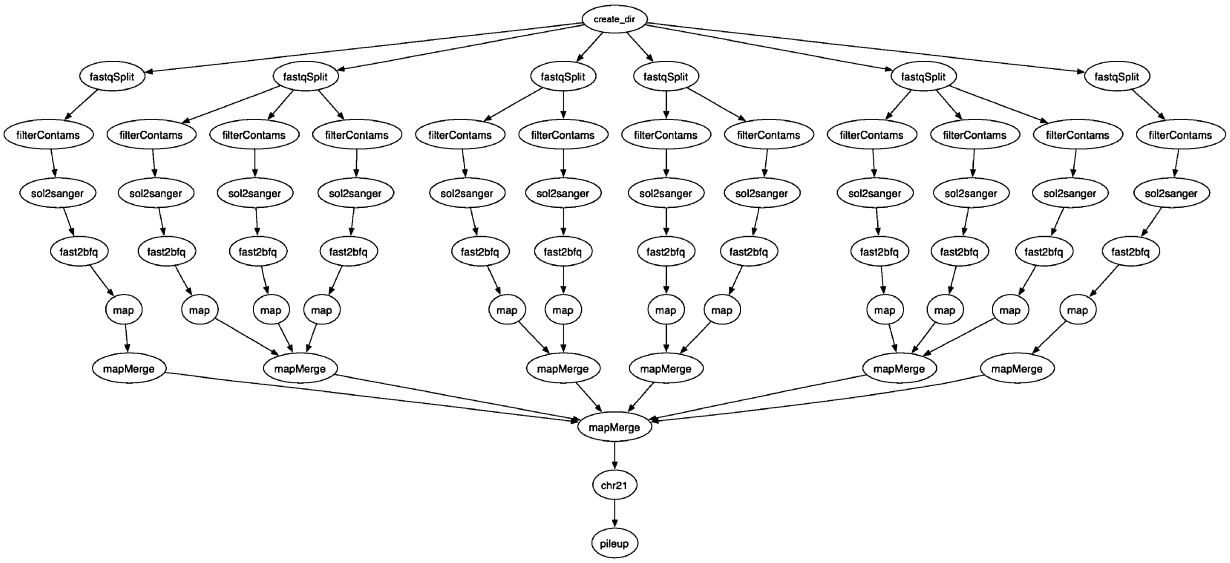

Fig. 9. View of a small, 64-task Epigenomic workflow.

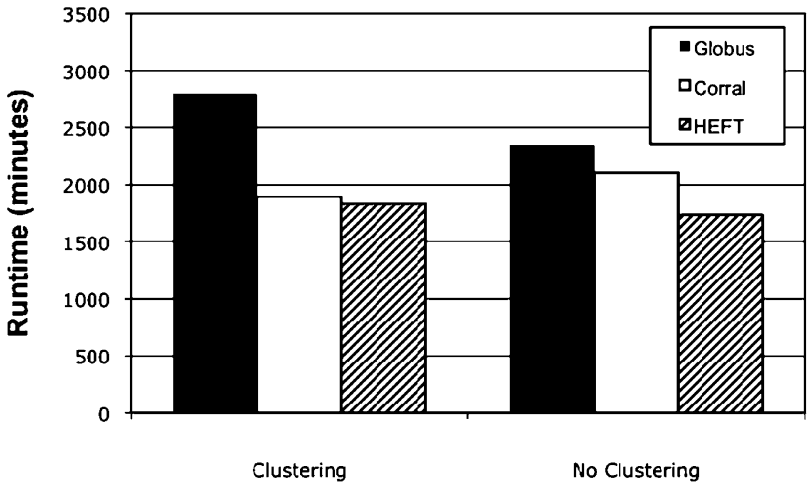

Fig. 10. Comparison of workflow runtime when limited by site policies.

head in the non-clustered case with Corral was only about 3% compared to HEFT, leaving the workflow unclustered is a reasonable solution.

It is important to note that although workflows using Globus are limited to 30 processors by HPCC's max_user_run policy, workflows using Corral are not. Corral can use a single parallel job to provision any number of processors. This allows Corral to allocate more processors to run the workflow than is possible using Globus. Figure 11 shows the runtime of the genomic workflow using 128, 256 and 512 processors. For this workflow 512 processors is the maximum parallelism achievable.

Using Corral to provision 512 processors resulted in an order of magnitude lower runtime (211 min) compared with the best runtime using Globus (2349 min). In addition, although the number of processors increases, the scheduling overhead when compared with HEFT remains relatively constant at around 5–10%.

4.3.4. Storage requirements

We analyzed the disk space required to run Condor glideins on a cluster. This includes the space required for executables, configuration files and logs. Because the size of these files varies depending on the runtime of the glideins (logs) and the architecture and system

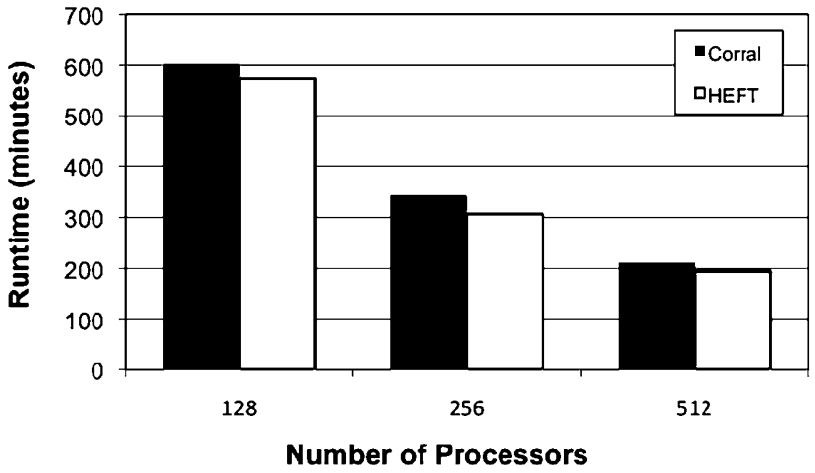

Fig. 11. Comparison of genomic workflow runtime on different numbers of processors.

Table 3
Sizes of files used by Condor glideins (in KB)

File type	Minimum size	Maximum size
Logs (per worker)	144	6144
Configuration (per worker)	5	5
Executables (per site)	20,480	44,032

Table 4
Storage required for multiple nodes (in MB)

Nodes	Minimum size	Maximum size
1	20	49
2	20	55
4	20	67
8	21	91
16	22	139
32	24	235
64	29	427
128	38	811

libraries of the cluster (executables), we report the minimum and maximum sizes that are possible (Table 3).

One important issue to note is that executables can be installed on a shared file system and used by multiple nodes, while logs and configuration files are generated for each node. Depending on the number of nodes allocated, the actual storage space used by the service may vary significantly. Table 4 shows the minimum and maximum amount of storage that would be required for multiple nodes.

In the worst case, the maximum space required for a pool of 128 nodes is 811 MB. This space is primarily consumed by Condor log files, which, by default, are allowed to grow up to 6 MB per node. However, because Corral automatically cleans up log files, and because the glideins will rarely be around long enough for log files to reach their maximum size, the actual size required will likely be closer to the minimum.

5. CyberShake

In this section we describe our experiences using Corral to provision resources for a large-scale workflow application on the TeraGrid.

The application, CyberShake [2,9], is a probabilistic seismic hazard analysis (PSHA) tool developed by the Southern California Earthquake Center (SCEC) to study the long-term risk associated with earthquakes. It consists of two parts: a parallel simulation that computes how a given geographic location, or site, responds to earthquakes, and a workflow that uses many scenario earthquakes to determine the future probability of different levels of shaking at the site.

The parallel simulation consists of an MPI code and several serial programs that generate a large 3D mesh of site-response vectors called Strain Green Tensors (SGTs). In this part of the computation there are only a few separate tasks and thus it can make use of standard grid scheduling techniques.

The workflow part of the computation is a 3-stage pipeline consisting of tasks that (1) extract site response vectors, (2) compute synthetic seismograms and (3) measure peak ground motions. The workflow contains ~840,000 tasks for each site. Due to the large number of tasks it is not feasible to use normal grid scheduling techniques because the overall workflow performance would be poor. Instead, Corral was used

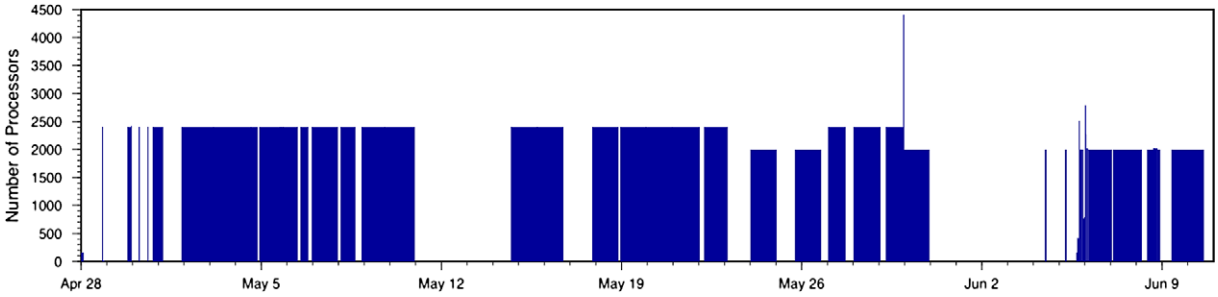

Fig. 12. CyberShake resource usage for the period between April 28, 2009 and June 11, 2009.

to provision resources and Condor was used to schedule the workflow tasks.

In the period between April 28, 2009 and June 11, 2009 the CyberShake team computed the seismic hazard for 220 sites located in Southern California. This work involved the execution of over 197 million workflow tasks. Corral was used to provision resources from the Ranger cluster at TACC. A timeline showing the resource usage for that period is shown in Fig. 12. The application made 64 provisioning requests for a total of 1.19 million CPU hours. The times where no resources were provisioned (white spaces in Fig. 12) correspond to the times when Ranger was down for maintenance or no new workflows were being scheduled and thus no provisioning requests were made. The figure also shows that for some time up to 4500 resources were provisioned. For most of the time 2400 cores were being provisioned for the workflow tasks.

Although in this case the provisioned resources targeted a single cluster, Corral is able to provision resources across a number of computational sites. In addition, a single Corral server can provision resources for multiple applications and multiple resource pools.

An open issue is how to optimize the resource utilization while running workflows on provisioned resources. It is possible for some fraction of the resources to sit idle while waiting for new workflow tasks. For large-scale workflows, such as SCEC, resource utilization can be improved by tuning the task scheduling system to release a sufficient number of tasks to the resources. For small workflows or workflows with a small number parallel tasks, a solution could be to run multiple, independent workflows or tune the number of resources provisioned over time.

After the execution of the 220 sites in the Spring of 2009, SCEC scientists are now analyzing the results of the computations and are preparing for an even larger set of runs in 2010.

6. Related work

Pilot-based resource provisioning systems have been used to acquire resources on Grid-based systems in order to overcome the job startup overheads and uncertainties. Here we mention the ones most related to our work and describe the differences between these systems and Corral.

Condor_glidein [7] is a command-line tool that can be used to add grid resources to an existing Condor pool using the glidein technique. Condor_glidein is simple to use, but unlike Corral it does not support advanced features such as an API, automatic resubmissions or the ability to provision multiple resources at once.

GlideinWMS [29] is a workload management system that is also based on Condor. It supports dynamic provisioning by polling Condor for queued application jobs and automatically provisioning grid resources to service them. Unlike Corral, which provides a direct interface for provisioning an exact number of resources for a definite period of time, glideinWMS automatically requests and releases resources for a group of users based on the current group workload.

MyCluster [38] creates personal clusters using several different resource managers including Condor. It can automatically maintain fixed-size pools by resubmitting resource requests as they expire, and it allows users to control the granularity of resource requests. It uses a virtual network overlay and user space network file system to avoid pre-staging executables. Unlike Corral, MyCluster does not have any programmatic interfaces that can be used to develop provisioning tools.

Falkon [27] is a multi-level scheduling system designed for applications requiring very high task throughput. It consists of a web service that accepts job requests, a provisioner that allocates resources from remote sites, and a custom node manager that executes application jobs on provisioned resources. Although

Table 5
A comparison of multi-level scheduling systems

System	Resource managers	User interfaces	Provisioning policies	Firewall negotiation	Resource provider interfaces
Condor_glidein	Condor	Command-line	Static	None	Globus
GlideinWMS	Condor	None (automatic)	Dynamic	GCB	Globus
MyCluster	Condor, SGE, OpenPBS	Command-line	Static	Manager on head node, virtual networking	Globus, PBS, SGE, LoadLeveler, LSF, Condor, EC2
Falkon	Custom	API	Static, dynamic	Manager on head node	Globus
VGES	Torque	API	Static	Manager on head node	Globus, EC2
DIANE	Custom	API, command-line	Static, dynamic	Outbound only	Ganga
Corral	Condor	API, command-line	Static	GCB, CCB	Globus

Falkon achieves very high throughput, it does so by omitting many of the features provided by off-the-shelf resource managers such as resource matching and job prioritization.

The VGES system includes a Java API that creates personal clusters on the grid [19]. The system uses a custom version of the Torque resource manager [37] that has been modified to run in user-mode. It creates personal clusters by starting Torque daemons on host clusters using grid protocols. Access to these personal clusters is provided through a user-level Globus gatekeeper that is started on the host cluster's head node. The system assumes that both Torque and Globus are installed and configured on the remote site and, unlike Corral, does not pre-stage executables.

DIANE [23] is a master-worker framework based on multi-level scheduling. It consists of a master process and several agent processes. The agents are started on grid worker nodes and contact the master to request tasks. The master distributes the tasks and merges the results. Like Falkon, DIANE relies on a custom scheduling system and does not support generic, off-the-shelf resource managers. And unlike Corral, DIANE requires applications to be developed using a custom API that ties the application strongly to the DIANE framework.

Table 5 summarizes the differences between the various systems. Clearly there is no system that has all the features, so there is a potential of developing more comprehensive solutions.

7. Conclusions

Scientists in many fields are developing large-scale, workflow applications for complex, data-intensive scientific analyses [34]. These applications require the use of large numbers of low-latency computational resources in order to produce results in a reasonable amount of time. Although the grid provides access to ample resources, the traditional approach to accessing these resources introduces many overheads and delays that make the grid an inefficient platform for executing workflows.

In this paper we presented the design and implementation of a resource provisioning system called Corral. Although Corral is a general-purpose resource provisioning system, it can greatly benefit the performance of workflow applications executing on clusters and the grid. The system is based on the concept of multi-level scheduling. This approach eliminates queuing delays by reserving resources, reduces overheads by streamlining resource management, and improves parallelism by allowing the user to specify application-specific scheduling policies.

We have shown how the use of Corral can improve the runtime of three real workflow applications. The system was shown to reduce the runtime of an astronomy application by 45% on average without clustering and 11% on average with clustering. Our results indicated that a combination of provisioning to reduce queue delays and clustering to amortize scheduling delays provided the best improvement in runtime. In addition, we showed how the system can be used to bypass restrictive site scheduling policies that, e.g. limit the number of processors that can be used concurrently. This enabled an order of magnitude reduction in the runtime of a genome mapping application.

Finally, we have shown that the system is being used today to enable the execution of scientifically meaningful workflows, such as those being run by earthquake scientists.

In the future we plan to expand the capabilities of the Corral system by adding support for parallel applica-

tion jobs, GSI security and dynamic provisioning. We are currently working to expand the backend capabilities of Corral to target other grid and cloud platforms, and we are working with the glideinWMS team [29] to use this expanded capability to provide a service-oriented front-end to glideinWMS.

Acknowledgements

This work was supported by the National Science Foundation under grants 0943725 (STCI-Provisioning) and 0749313 (PetaShake). The authors would like to thank the SCEC researchers, Scott Callaghan, Patrick Small, Kevin Milner, and Phil Maechling, for information about CyberShake and its execution on the TeraGrid.

References

[1] J. Blythe, S. Jain, E. Deelman, Y. Gil, K. Vahi, A. Mandal and K. Kennedy, Task scheduling strategies for workflow-based applications in grids, in: *5th IEEE International Symposium on Cluster Computing and the Grid (CCGrid '05)*, Vol. 2, Cardiff, UK, 2005, pp. 759–767.

[2] S. Callaghan, P. Maechling, E. Deelman, K. Vahi, G. Mehta, G. Juve, K. Milner, R. Graves, E. Field, D. Okaya and 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)*, Indianapolis, IN, 2008.

[3] H. Casanova, A. Legrand, D. Zagorodnov and F. Berman, Heuristics for scheduling parameter sweep applications in grid environments, in: *9th Heterogeneous Computing Workshop*, Cancun, Mexico, 2000, pp. 349–363.

[4] A. Chervenak, N. Palavalli, S. Bharathi, C. Kesselman and R. Schwartzkopf, Performance and scalability of a replica location service, in: *13th IEEE International Symposium on High-Performance Distributed Computing (HPDC '04)*, Honolulu, HI, 2004, pp. 182–191.

[5] A. Chervenak, R. Schuler, M. Ripeanu, M.A. Amer, S. Bharathi, I. Foster, A. Iamnitchi and C. Kesselman, The globus replica location service: design and experience, *IEEE Transactions on Parallel and Distributed Systems* **20** (2005), 1260–1272.

[6] Condor Connection Brokering (CCB), available at: http://www.cs.wisc.edu/condor/manual/v7.3/3_7Networking_includes.html#SECTION00473000000000000000.

[7] Condor_glidein, available at: http://www.cs.wisc.edu/condor/glidein.

[8] E. Deelman, Grids and clouds: making workflow applications work in heterogeneous distributed environments, *International Journal of High Performance Computing Applications*, December 4, 2009, available at: http://hpc.sagepub.com/cgi/content/abstract/1094342009356432v1.

[9] E. Deelman, S. Callaghan, E. Field, H. Francoeur, R. Graves, N. Gupta, V. Gupta, T.H. Jordan, C. Kesselman, P. Maechling, J. Mehringer, G. Mehta, D. Okaya, K. Vahi and L. Zhao, Managing large-scale workflow execution from resource provisioning to provenance tracking: the CyberShake example, in: *2nd IEEE International Conference on e-Science and Grid Computing (e-Science 06)*, Amsterdam, The Netherlands, 2006.

[10] E. Deelman, C. Kesselman, G. Mehta, L. Meshkat, L. Pearlman, K. Blackburn, P. Ehrens, A. Lazzarini, R. Williams and S. Koranda, GriPhyN and LIGO, building a virtual data grid for gravitational wave scientists, in: *11th IEEE International Symposium on High Performance Distributed Computing (HPDC '02)*, Edinburgh, Scotland, UK, 2002, pp. 225–234.

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

[12] R. Fielding, Architectural styles and the design of network-based software architectures, PhD thesis, University of California, Irvine, 2000.

[13] J. Frey, T. Tannenbaum, M. Livny, I. Foster and S. Tuecke, Condor-G: A computation management agent for multi-institutional grids, *Cluster Computing* **5** (2002), 237–246.

[14] Generic Connection Brokering (GCB), available at: http://cs.wisc.edu/condor/gcb.

[15] W. Gentzsch, Sun Grid Engine: towards creating a compute power grid, in: *1st IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid '01)*, Brisbane, Australia, 2001, pp. 35–36.

[16] G. Juve and E. Deelman, Resource provisioning options for large-scale scientific workflows, in: *IEEE Fourth International Conference on e-Science (e-Science 08)*, Indianapolis, IN, 2008, pp. 608–613.

[17] G. Juve, E. Deelman, K. Vahi and G. Mehta, Scientific workflow applications on Amazon EC2, in: *Workshop on Cloud-Based Services and Applications in Conjunction with 5th IEEE International Conference on e-Science (e-Science 2009)*, Oxford, UK, 2009.

[18] D.S. Katz, J.C. Jacob, E. Deelman, C. Kesselman, S. Gurmeet, S. Mei-Hui, G.B. Berriman, J. Good, A.C. Laity and T.A. Prince, A comparison of two methods for building astronomical image mosaics on a grid, in: *34th International Conference on Parallel Processing Workshops (ICPP '05 Workshops)*, Oslo, Norway, 2005, pp. 85–94.

[19] Y. Kee, C. Kesselman, D. Nurmi and R. Wolski, Enabling personal clusters on demand for batch resources using commodity software, in: *International Heterogeneity Computing Workshop (HCW 08) in Conjunction with IEEE IPDPS 08*, Miami, FL, 2008.

[20] M.J. Litzkow, M. Livny and M.W. Mutka, Condor: A hunter of idle workstations, in: *8th International Conference of Distributed Computing Systems (ICDCS '88)*, San Jose, CA, 1988, pp. 104–111.

[21] M. Maheswaran, S. Ali, H.J. Siegel, D. Hensgen and R.F. Freund, Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems, in: *8th*

Heterogeneous Computing Workshop, San Juan, Puerto Rico, 1999.

[22] Maui Cluster Scheduler, available at: http://www.supercluster.org/maui.

[23] J. Moscicki, DIANE – distributed analysis environment for GRID-enabled simulation and analysis of physics data, in: *IEEE Nuclear Science Symposium Conference Record*, Portland, OR, 2003, pp. 1617–1620.

[24] D. Nurmi, R. Wolski and J. Brevik, VARQ: virtual advance reservations for queues, in: *17th International Symposium on High Performance Distributed Computing (HPDC '08)*, Boston, MA, 2008.

[25] Open Science Grid, available at: http://www.opensciencegrid.org.

[26] PBSPro, available at: http://www.pbspro.com.

[27] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster and M. Wilde, Falkon: a Fast and Light-weight tasK executiON framework, in: *Supercomputing 2007*, Reno, NV, 2007.

[28] SDSC User Portal, available at: http://portal.sdsc.edu.

[29] I. Sfiligoi, GlideinWMS – a generic pilot-based workload management system, *Journal of Physics: Conference Series* **119** (2008), 062044.

[30] G. Singh, C. Kesselman and E. Deelman, Optimizing grid-based workflow execution, *Journal of Grid Computing* **3** (2005), 201–219.

[31] G. Singh, C. Kesselman and E. Deelman, Performance impact of resource provisioning on workflows, Technical Report 05-850, University of Southern California, 2005.

[32] G. Singh, M. Su, K. Vahi, E. Deelman, B. Berriman, J. Good, D.S. Katz and G. Mehta, Workflow task clustering for best effort systems with Pegasus, in: *15th ACM Mardi Gras Conference*, Baton Rouge, LA, 2008.

[33] Southern California Earthquake Center (SCEC), available at: http://www.scec.org.

[34] I.J. Taylor, E. Deelman, D.B. Gannon and M. Shields, *Workflows for e-Science: Scientific Workflows for Grids*, Springer-Verlag, New York, 2006.

[35] TeraGrid, available at: http://www.teragrid.org.

[36] H. Topcuoglu, S. Hariri and W. Min-You, Performance-effective and low-complexity task scheduling for heterogeneous computing, *IEEE Transactions on Parallel and Distributed Systems* **13** (2002), 260–274.

[37] Torque, available at: http://supercluster.org/torque.

[38] E. Walker, J.P. Gardner, V. Litvin and E. Turner, Creating personal adaptive clusters for managing scientific jobs in a distributed computing environment, in: *IEEE International Workshop on Challenges of Large Applications in Distributed Environments (CLADE 06)*, Paris, France, 2006.