International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:7, 2009

SWARM: A meta-scheduler to Minimize Job Queuing Times on Computational Grids

Jean-Alain Grunchec, Jules Hernández-Sánchez, and Sara Knott

***Abstract*—**Some meta-schedulers query the information system of individual supercomputers in order to submit jobs to the least busy supercomputer on a computational Grid. However, this information can become outdated by the time a job starts due to changes in scheduling priorities. The MSR scheme is based on Multiple Simultaneous Requests and can take advantage of opportunities resulting from these priorities changes. This paper presents the SWARM meta-scheduler, which can speed up the execution of large sets of tasks by minimizing the job queuing time through the submission of multiple requests. Performance tests have shown that this new meta-scheduler is faster than an implementation of the MSR scheme and the gLite meta-scheduler. SWARM has been used through the GridQTL project beta-testing portal during the past year. Statistics are provided for this usage and demonstrate its capacity to achieve reliably a substantial reduction of the execution time in production conditions.

***Keywords*—**Grid Computing, Multiple Simultaneous Requests, Fault tolerance, GridQTL.

I. Introduction

IN recent years, the Grid has emerged as one of the most prominent solutions in supercomputing. Computational Grids enable the coordination of large computational resources in a geographically distributed environment across multiple administrative domains [1]. Setting up a Grid of High Performance Clusters (HPCs) scattered across several organizations requires sets of software tools called middleware such as the Globus Toolkit (GT) [2]. In order to submit jobs to Grid resources with the Globus Toolkit, users need to set up a certificate on the file system of the submitting machine and must authenticate through the Grid Security Infrastructure (GSI). When several people belong to the same research group, it may be convenient to set up a web portal which allows users to access a certain type of application. For security reasons, a single certificate is then used and can only be accessed by the administrator of the web server.

J.-A. Grunchec is with the Institute of Evolutionary Biological Sciences, School of Biological Sciences, University of Edinburgh, EH9 3JR UK. Phone number: +44 (0)131-650-5442; fax number: +44 (0)131-650-6556; e-mail: jgrunche@staffmail.ed.ac.uk.

J. Hernández-Sánchez is with the Institute of Evolutionary Biological Sciences, School of Biological Sciences, University of Edinburgh, EH9 3JR UK Phone number: +44 0131-650-5442; e-mail: jules.hernandez@ed.ac.uk).

S. Knott is with the Institute of Evolutionary Biological Sciences, School of Biological Sciences, University of Edinburgh, EH9 3JR UK. Phone number: +44 0131-650-5444; e-mail: s.knott@ed.ac.uk .

A GridSphere [3] portal was created for the GridQTL project [4] to provide tools for geneticists to perform Quantitative Trait Loci (QTL) mapping analyses [5] on the Grid. QTL are chromosomal regions affecting trait variation, and therefore likely to harbor genes of interest. Most GridQTL computations can be distributed on the Grid as Parameter Sweep Applications (PSA) i.e. computations that can be divided into independent tasks identified by a unique combination of parameters. Typically, a region of interest (e.g. part of a chromosome or the whole genome) is tested at regular intervals for the presence of QTL. A new GridQTL tool is the Linkage-Disequilibrium and Linkage Analysis (LDLA) module. It deals with complex populations, with the additional advantage that it incorporates information from pedigree and history simultaneously. LDLA has been used mainly to refine QTL location [6], implying analyzing over 10 to 200 positions in the region of interest. Each position can be analyzed independently by a single task. LDLA uses likelihood optimization software in order to calculate the variance of QTL and the duration of the calculation depends on the number of steps to reach convergence.

Tasks within one LDLA share input data. Therefore, the input dataset can be co-located i.e. it can be shared by all individual tasks. It is expected to be less than a few megabytes. GridQTL uses the NGS [7] as the main computational tool. GridQTL computations can also be sent to a local HPC, the ECDF [8] and a small local pool of workstations managed through the Condor middleware [9]. The computing elements of this Grid can be considered as quite homogenous since the slowest computing unit on the Condor pool can be roughly assessed as half the speed of the fastest element in the ECDF. The duration of the scheduled tasks depends on the type of data, the type of analysis and the number of steps necessary for the convergence of the likelihood calculation. Any assessment of the expected duration of each task is therefore expected to be quite inaccurate. The aim is to minimize the jobs *makespan* i.e. the time between sending the first input files to a computational server and receiving the last output. Since file transfer durations are considered negligible for LDL analyses when compared with the queuing and execution times and since the execution duration forecasts are expected to be inaccurate, our optimization effort has chiefly consisted in trying to minimize the time spent by these tasks in the queuing system of the Grid. This led us to develop a meta-scheduler called SWARM (Scheduling With A Request Multiplication) which

substantially speeds up PSAs by minimizing the queuing time on the Grid, and provides facilities required in a production environment such as failure handling and dynamic activity reports through a web interface.

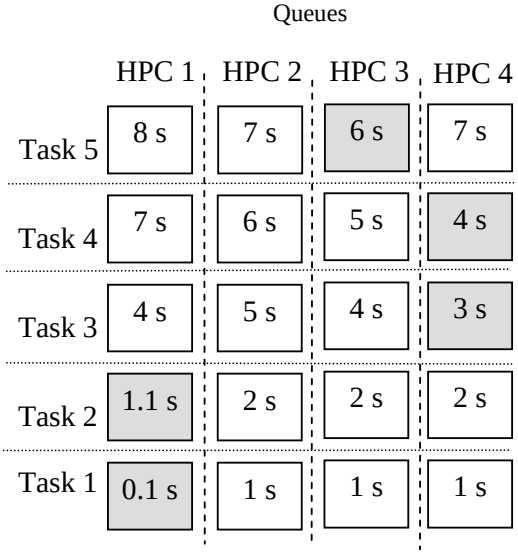
Section 2 describes the details of the SWARM system architecture. In Section 3, the performance of the scheduler is evaluated. Section 4 presents some related schedulers, summarizes the findings and presents possible extensions.

II. METHODS AND MATERIALS

A. The Multiple Simultaneous Request (MSR) algorithm

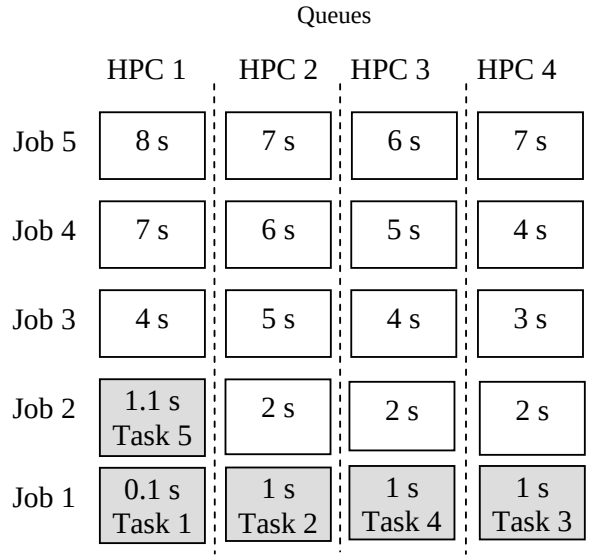
A task is defined as the calculation, e.g. computing a likelihood, and a job can be described as the means to perform the task on the Grid. When a task needs to be executed, it would intuitively make sense to schedule a job on the fastest cluster on the Grid with available nodes. However, querying the clusters on a Grid can take a few seconds due to the execution duration of the calls and the latency of the communications. By the time a meta-scheduler can access the information about the CPU availability for each cluster belonging to the Grid, this information may already have become inaccurate due to new scheduling events happening elsewhere in the meantime. This problem is compounded by the fact that a Grid includes many clusters and faults in the information system of any cluster should be expected at any time. When no cluster has any free node available, it is still possible to check that some clusters have queued jobs with fewer computational requirement than others by querying the workload management system of each cluster by means of middleware commands. Nevertheless, if a high priority user submits a job B after job A has been submitted, job A may have to wait in the queue until job B has started. Consequently, since the information available at the time of submission about the status of a particular cluster may change, it may be better to submit a job to several clusters in order to increase the chance of a short queuing duration for a particular task.

The Multiple Simultaneous Requests (MSR) job algorithm has been designed to improve the performance when the priorities scheduled in supercomputers change [10]. In the original description, for each task which needs to be processed, a job is submitted on the *k* least loaded clusters. The first of these *k* jobs to start will carry out the task while the others will be cancelled, hence the MSR minimizes the queuing duration across these k clusters. Each job runs a call to the central meta-scheduler and checks if the task has started elsewhere. The operation is supposed to be atomic. However, the cancellation calls may be processed after some of these *(k-1)* replicates (from now on referred to as "redundant replicates") have started, and this can waste some CPU time. On a Grid composed of a few large clusters, *k* can be chosen as the number of clusters so that this step is not required.



The grey boxes correspond to the time of the start of a particular task on a particular cluster. The white boxes correspond to the time of the cancellation of the redundant jobs associated with a particular task. The times are given with hypothetical values in seconds since the submission of the first job of the analysis i. e. the queuing times.

Fig. 1 Time of the start and cancellation calls of jobs associated with a 5 tasks analysis on a 4 HPC Grid with the MSR algorithm



The grey boxes correspond to the time of the start of a particular task on a particular cluster. The white boxes correspond to the time of the cancellation of the redundant jobs associated with a particular task. The times are given with hypothetical values in seconds since the submission of the first job of the analysis i. e. the queuing times.

Fig. 2 Time of the start and cancellation calls of jobs associated with a 5 tasks analysis on a 4 HPC Grid with the SWARM algorithm.

The MSR method has substantial theoretical advantages over more conventional methods since it is expected to deal more robustly with information system failures through the use of multiple requests. The original paper [10] presented test

results of the MSR scheme on the simulation of the activity of a 5000 contiguous jobs subset from the Cornell Theory Center. However, no implementation has been developed and released to work in a real grid environment. Hence, the MSR scheme did not take into account issues which happen in real production environments, such as network or node failures.

B. The SWARM algorithm

In a PSA context where *n* tasks need to be carried out, it would be possible to submit *n* MSR analyses and bundle the results at the end. That would typically mean $n \times k$ jobs if k is the number of clusters used on the Grid by the MSR. Unfortunately, it is likely that many of the redundant replicates of the first tasks will run and be cancelled before the later tasks have started. For instance, Fig. 1 describes the theoretical case of 5 tasks scheduled with the MSR algorithm on a Grid made of four clusters. Each task is expected to run for less than a second, i.e. a duration that is smaller than durations observed in reality between the scheduling decisions. In this example, 10 redundant jobs ran somewhere on the Grid before the last job corresponding to Task 5 started 6 seconds after the jobs submission on HPC 3. The SWARM algorithm attempts to increase the overall speed by allowing redundant replicates of the first task to act as potential replicates of the later tasks (Fig. 2). Therefore no redundant replicates can happen before the last task has started. In the MSR, the replicates try to check if a job has started elsewhere for their respective task, whereas in the SWARM scheduler, the tasks send call-backs to the server to check the first index of the task which has not yet been processed.

C. Implementation

Next, the relation between the tasks executed on the Grid is described. Java Servlets [11] can communicate with remote clients provided the firewall on these remote clients allows outbound HTTP calls. Most NGS HPCs allow either the nodes to communicate with machines beyond the firewall through HTTP or indirectly via the head node. Consequently, a servlet has therefore been chosen as the corner stone of the SWARM meta-scheduler.

The analysis starts when a web client calls the SWARM servlet and passes the information specifying the type of analysis and the parameters needed by the analysis. The input files that are common to all tasks are then uploaded on the head node of each Globus cluster. A Submit Description File is generated for each Condor pool so that the transfer of the common input files is done after the job submission. *n* jobs are then submitted to each cluster. For each of these jobs, a *maximal job duration* is calculated.

Once the jobs are submitted, they will enter the queue on each cluster. When they start, they contact the SWARM servlet and wait for the servlet to send back the index *i* of the next task due to start. Input files that are required by a specific task only are then sent through GridFTP to the Globus resources. If this task runs on a Condor resource, the specific input file can also be downloaded from the web server.

When a task has been successfully executed, the associated job contacts the SWARM servlet, which then retrieves the results from the Globus resources. The result files are then parsed. If they do not correspond to an expected format, it can be assumed that there was some kind of failure during the job execution (either a corruption of data or problems with the node executing the job). This type of failure is considered as an *output failure*. If the job has failed, the job is deleted from the memory, and therefore its task index can be allocated to another job that will start later. In case of output failure, new jobs are submitted to all clusters in order to ensure that a failed job can be restarted up to three times per task. If the job is considered as successful, then the completing process checks the number of other jobs having successfully completed. If all scheduled tasks have completed successfully, each cluster needs to cancel the redundant jobs.

SWARM has been developed to deliver a service to a community of users running hundreds of analyses and thousands of tasks per day. Consequently, it must handle different types of failure. When the execution duration of a job on the Grid exceeds its expected maximal duration, the job is likely to be terminated by the cluster before the job ends successfully. In this case, a *watchdog thread* cancels the job and schedules it again with an expected maximal duration multiplied by three. This will be done for up to three *duration failures* per job. Jobs can also fail on the Grid due to hardware failures. In this case, the jobs do not report their results back to the SWARM servlet, and are then considered to have terminated due to a *silent failure*. In order to deal with computing node crashes, the watchdog thread checks that tasks allegedly active according to the servlet records are indeed active through Globus and Condor status checking commands. If the job is inactive and result files are missing, new jobs are submitted in the same way as in the case of output failures. In the case of a complete shutdown of a large part of the network around the server during the analysis submission, many job submissions may fail, since the clusters can not be contacted. If the watchdog thread notices that there is no active job on the Grid for a long time, it will attempt to submit new jobs on each active cluster.

As described above, several threads can be running on the server while they try to check and alter the status of the jobs. In order to guarantee thread safety, Java synchronized statement are extensively used to protect the integrity of the data. In order to increase performance, reports retrievals are made without synchronization when an analysis is still running.

D. Experimental Protocol

In order to assess the performance of SWARM, a set of tests were run to compare the SWARM implementation, the MSR implementation and a third party meta-scheduler, the gLite Resource Broker (RB) [12]. gLite [13] is a middleware that has been developed as part of the EGEE project [14]. EGEE provides access to 41,000 CPUs across 250 sites worldwide.

The tests consisted of mapping QTL in a simulated population of 400 individuals. The same dataset was used for each test. Every test consisted of running 50 LDL Analyses at different positions simultaneously, so for each test 50 tasks had to be executed on the Grid. Each task lasted for about 2 minutes and the total duration for each test is about two hours. These durations actually depend on the HPCs used during the computation. In terms of genetics, each analysis aimed to locate a simulated gene within a short chromosomal region. The chosen set-up resembles typical conditions that have been encountered in real data analyses.

A series of simultaneous tests was run. Three sets of 15 tests were carried out: one set with SWARM, another one with the MSR and a last one with gLite. For each subset, a couple of tests were run each day, with a working day considered as starting and finishing at 09:00 hrs. This allowed us to test the schedulers at different times, in different load conditions. For each test, the gLite RB, the MSR scheme and the SWARM scheduler are started within a 5 s time interval in this order. During these tests, the three schedulers are expected to compete simultaneously to complete their tasks in the shortest duration.

The resources used for each set of tests vary depending on the availability of the resources for each scheduler and the type of tests. The GridQTL project can access a set of NGS resources, a local Condor pool and the ECDF located in Edinburgh (Table I). These resources have different CPU specifications. The seven computational resources are available to both the SWARM and MSR schedulers. The gLite RB has been deployed in pre-production in March 2007 on the NGS. It runs on the NGS Oxford, Leeds, RAL and Westminster HPCs. At the time of the experiments, the NGS Manchester HPC could not be used by the gLite RB, and gLite had yet to be set up on the ECDF. Consequently, the HPC used for the tests are the four supercomputers that are available to the gLite RB.

Before the test could be run, the LDLA had first to be implemented with the different schedulers. The SWARM tests were implemented as single SWARM analyses of the 50 positions. It is possible to narrow the SWARM algorithm so that it runs the MSR scheme. The MSR tests have been implemented as 50 SWARM analyses submitted and run in parallel. Each MSR analysis is expected to handle a single position. The steering algorithm works as follows: since all the 50 analyses share their input data, it is transferred before the job submission with the meta-scheduler. Then, the MSR analyses submission starts through a set of calls to the meta-scheduler. Once all MSR tasks have been submitted, the script calls the meta-scheduler every 5 seconds and checks the status of the 50 MSR analyses at once. When the 50 MSR analyses have completed, the resulting output files are post-processed. Then, the MSR test has terminated and the test duration and statistics are recorded.

For the gLite tests, a feature of gLite allows a ranking to be set between the machines to be selected for job submission. The ranking method for site selection has been set so that sites with the most available CPUs are chosen in priority if no job is queuing there. If all sites have some queuing jobs, the site selected is the site with the smallest number of jobs in the queue. This ranking method is supposed to be better than only using the number of free CPUs according to gLite documentation since the number of free CPUs is advertised per queue and not per virtual organization [15]. The gLite implementation uses an algorithm similar to the MSR implementation. First, the input files are sent simultaneously to each of the sites. When all transfers have terminated, the shell script submits 50 jobs with the gLite RB. When all analyses have been submitted, a loop is started and the script checks if a job has completed during every iteration. If a job has completed, the results are downloaded and then the loop moves on to the next iteration. If the current job has not yet completed, the scripts sleeps for 5 s, and checks again whether the job has completed, and so on until completion. Once the computation has completed, the result file for each gLite job are post-processed, and then duration statistics are recorded. It is important to note that this implementation does not include features for failures management such as restarting failed jobs. Consequently, when a gLite test stopped because of a failure, the entire test was discarded and a new one was started instead.

TABLE I
RESOURCES AVAILABLE TO THE SWARM SCHEDULER

Cluster	NbCPU	T100MF	WMS	Used during the tests
ECDF	1536	0.67	SGE	NO
NGS-2 Westminster	256	0.99	PBS	YES
NGS-2 Oxford	256	1.03	PBS	YES
NGS-2 Leeds	256	0.97	PBS	YES
NGS-2 RAL	256	1.05	LSF	YES
NGS-2 Manchester	256	1.08	PBS	NO
Local Condor Pool	6	Min : 0.64 Max : 0.65	Condor	NO

NbCPU is the total number of cores of the compute nodes.

T100MF is the time taken by a CPU to run the 100 MFlops in a second according to our benchmark, which is a C++ program.

WMS is the the Workload Management System available on the HPC (SGE = Sun Grid Engine, PBS = Portable Batch Scheduler, LSF = Load Sharing Facility).

III. RESULTS

A. Experimental performance measurements

Several measurements were recorded during these tests. Table II provide results about the wall clock duration, the real duration, the speed up ratio, the total duration of the redundant replicates, the number of output failures, the queuing time and the number of HPC used successfully during the computation. As for all durations in these experiments, the time unit used is the second. The real duration is the duration between the start of the process calling the LDLA script and the termination of the last output files retrieval from the compute nodes. The wall clock duration is defined as the total sum of the durations of the successful jobs running on the Grid. The speed up ratio

is calculated as the wall clock duration divided by the real duration. The "queuing time" for a particular job is defined as the duration between start of the process calling the LDLA script for each of the scheduler and the start of the job on the compute node. The queuing time includes the duration of the transfer of the common input files to the head nodes of the HPCs. The "redundant duration" is defined as the total sum of the wall clock durations of the redundant replicates running on the Grid. In the case of the MSR and SWARM experiments, the number of output failures was recorded. There were also silent failures but these were not recorded as they do not require jobs to be restarted if they are not also output failures i.e. when a job runs successfully but does not report back as completed. Except in a few cases, the output failures were temporary failures i.e. the results were obtained after the failed jobs were re-submitted. The number of HPCs used during the computation is the number of HPCs on which at least one job ran, successfully completed and gave results back to the cluster.

The simultaneous series of tests aimed to provide an environment where the load conditions and computing nodes were the same (Table II). For all tests except one, the speed up ratio is higher for SWARM and the MSR than for the gLite RB.

The MSR and SWARM scheduler are significantly faster than the gLite resource broker for this set of analyses In the case of the MSR scheme, the average real duration, the average wall clock duration and the speed up ratio are respectively 628.8 s, 8110 s and 15.8. SWARM is even faster since the average real duration equals 524.6 s, the wall clock duration is 8612 s and the speed up ratio equals 21.4. For 11 tests, SWARM is the fastest scheduler to complete the computation, whereas the MSR is faster in 3 other cases. This can be related to shorter queuing times for SWARM since the average queuing time equals 276.9 s in the case of the MSR whereas the average queuing time is 143.6 s for SWARM. Similarly, the maximum queuing time 457.6 s for the MSR is longer than the maximum queuing time 347.8 s for SWARM. The average number of output failure for the MSR was 1.6 and 1.7 for SWARM. Since no resumption mechanism has been implemented in gLite, output failures were not recorded and in case of failures another test was performed.

B. SWARM as a production scheduler

SWARM has been used since March 2008 as part of the pre-production LDLA beta-testing portlet [16]. This portlet has been developed within the scope of the GridQTL [4] project. GridQTL uses an open source portlet-based portal framework known as GridSphere [3]. GridSphere portlets are self-contained modular software components that provide specific facilities within a general portal architecture. They typically appear as pluggable windows on the web page, and provide a flexible and intuitive interface to a web portal. Both the GridSphere portlets and the SWARM meta-scheduler use the Apache Tomcat web container [17]. It is therefore quite straightforward to have the GridSphere portlets work alongside SWARM. In order to provide usage transparency, a technology called AJAX [18] allows dynamic calls from a web page to a servlet running on a server. This method combines Asynchronous JavaScript calls and XML HTTP requests. A JavaScript timer can refresh the web page at a given frequency and hence provides a dynamic interface. XML HTTP requests to a web server allow the text provided within an HTTP response to be used in a JavaScript function.

TABLE II
PERFORMANCE MEASUREMENTS OF SWARM, MSR & GLITE

Type	Queuing Time (s)		Duration (s)			Speed up ratio
	Avg.	Max.	Real	Wall Clock	Redun-dant	
gLite	359	731	3475	9878	n.a.	5.6
MSR	257	458	629	8110	28	15.8
SWARM	144	348	525	8612	37	21.4

The values given correspond to the average values of the measurements over the corresponding 15 tests, without taking into account the single interrupted gLite test.

Previous sections presented tests involving a LDLA with 50 positions to be analyzed. Some of the users have run much larger computations, and this has helped us to address some weaknesses in the scheduler. For instance, some users ran LDLA analysis on hundreds of positions. Intuitively, it would be most efficient to send the final cancelling calls all together in parallel in order to cancel many redundant replicates before they start and therefore reduce the redundant duration. However, it has been noticed that sending hundreds of cancelling calls simultaneously had severe side effects on the server's performance, and could crash the Tomcat server. Therefore thread pools are used to send a certain, limited maximum number of simultaneous cancelling calls. For most analyses run in the LDLA beta-testing phase, it happened rarely that a single cluster processed more than 70 percent of the tasks. Consequently, other improvements were implemented to reduce the number of redundant replicates. For instance, the configuration file allows a reduction in the number of replicates on a particular cluster.

When the MSR scheme can access both NGS resources, the Condor pool and the ECDF, some tests have shown that the MSR scheme seems to often select the ECDF, the fastest HPC according to Table I, and although this choice should lead to smaller computing times, MSR was still slower than SWARM. A simple feature has therefore been added to SWARM for production use to favor the selection of faster clusters if they are available: it is possible to set in the configuration files a small delay before the submission of jobs on particular clusters. This is to help faster clusters to be selected when several HPCs of heterogeneous speed are available. The slower the HPCs, the longer the duration of this delay can be set.

A computational grid can be heterogeneous in terms of the speed of the HPCs involved, but it is chiefly the heterogeneity of the policies of use between virtual organizations that differentiates Grid computing from distributed computing. In particular, the NGS restricts its usage by providing a limited

CPU hours account whereas access to the ECDF and the Condor pool is not restricted. In order to improve the level of service provided to the users, the SWARM scheduler can check a monthly CPU allowance provided to each user and submit jobs on a set of resources depending on the previous usage and therefore implement a fair share usage policy among users.

Since its release, the use of SWARM through the LDLA portlet has enabled bug identification and correction and the testing of the capacity of SWARM to significantly speed up much larger computations than those run in section 3. Between March 2008 and April 2009, SWARM has allowed 57 external users to run 1934 analyses i.e. a total of 155684 independent tasks, which required 10118 hours of computation. Table III displays the 15 largest computations run in October and November 2008. They show the capacity of SWARM to achieve large speed up ratios of up to 147 on computations lasting for more than 28 hours.

Some datasets uploaded by the LDLA external users required tens of gigabytes of RAM and tens of gigabytes of temporary space storage on the HPC head nodes in order to be analyzed. All clusters cannot provide this to all jobs. Some features were added to SWARM so that entire HPC computing nodes can be booked for particular jobs requiring a lot of memory. When some analyses require a large amount of temporary disk space, the supercomputers allow only a few jobs to run on them depending on the size of their local storage facilities and the disk size required by the job. This may lead to situations where there are fewer jobs scheduled on the Grid than tasks to be performed. In this case, the resubmission mechanism of SWARM is then used to submit new jobs when the previous ones have terminated and the temporary space storage has been made available again. This mechanism will be called until all tasks have been processed.

TABLE III
THE 15 LARGEST LDLA COMPUTATIONS RUN BY 6 EXTERNAL USERS IN OCTOBER AND NOVEMBER 2008

Time and date of the computation	Real duration in minutes	Wall clock duration in hours	Speed up ratio
13/10 09:50	74.1	49.6	40.2
27/10 09:08	55.7	40.4	43.5
27/10 10:45	53.1	39.7	44.9
30/10 09:29	13.9	29.5	127.0
31/10 02:51	112.5	37.2	19.9
31/10 06:22	119.6	36.4	18.2
01/11 13:37	14.8	36.5	147.8
05/11 10:49	14.3	33.7	141.8
05/11 11:19	18.5	35.4	114.6
08/11 01:20	190.9	286.5	90.0
08/11 09:52	370.4	106.9	17.3
11/11 16:45	353.0	171.7	29.2
11/18 16:49	74.4	46.0	37.1
11/19 08:57	14.4	31.0	129.2
11/19 13:27	93.0	47.4	30.6

IV. DISCUSSION

Scheduling jobs effectively is one of the most crucial problems in Grid computing. Some middleware try to provide a solution by focusing on particular types of applications, e.g. Nimrod/G [19] and AppLeS [20], whereas Condor [9] is particularly good at handling a large number of jobs on a set of resources in a distributed computing context and has been extended to be also an all-around solution in Grid computing.

Nimrod/G's scheduling approach is based on deadlines and on a Grid economy model [19][21]. Some input parameters need to be provided. For instance, a resource owner has to set the *resource cost*. The user has to specify the maximal *price* to be paid and may provide a *deadline* before which an application execution needs to have completed. The Nimrod/G resource broker queries the Globus Monitoring and Discovery System (MDS) and identifies available resources. Then, it tries to distribute jobs so that the maximal usage cost, which depends on the usage cost of each resource, and deadlines are respected. During the execution, when cheaper or faster available resources are discovered, jobs may be migrated away from slow or expensive resources. Nimrod/G can evaluate the job duration by using some execution rate measurement. The Gridbus Resource broker provides an extension to Nimrod/G to optimize PSAs with large datasets [22]. Features were included in order to enable trading the capacity to speed up a distributed calculation versus a possible reduction of the economical cost of the calculation on a world wide Grid [23]. A portal for high energy physics and astrophysical computations was designed to steer the calculations with the Gridbus broker on Grids [24]. Grid activity progress reports can be generated at the user's request though there is no facility such as AJAX to automate reports delivery. This portal has been designed so that it can be easily extended to cope with different types of physics computations.

AppLeS [20] is another meta-scheduler that focuses on PSAs and more particularly on co-location of data and experiments and adaptive scheduling. Basically, AppLeS tries to minimize dynamically the job *makespan*. AppLeS monitors the state of the Grid by frequently calling *scheduling events*. For each scheduling event, a heuristic is used in order to provide the best scheduling plan and requires only the estimation of the execution and transfer time. Both AppLeS and Nimrod/G continuously re-evaluate and plan again their schedule: they frequently use scheduling events during which they query resources about their status, identify available resources and then use their own heuristics to create a scheduling plan with the available resources. The AppLeS scheduler can also be used through a web portal. For instance, AppLeS is used in the GridSpeed [25] project, which allows non-specialists to develop application portals.

Condor [9] is a general purpose middleware for Grid and distributed computing. Condor allows a cluster to be created that collects the computing power of thousands of idle workstations. A single job queue is created on the Condor Central manager, and as soon as a resource becomes available, the job with the highest priority can be scheduled on this resource. This central manager keeps a record of the activity of each resource, and is responsible for migrating jobs. A mechanism of Condor called Condor-G allows jobs to be submitted to Globus resources [26]. Another Condor

mechanism called *GlideIn* transforms Globus resources into resources that can be accessed directly through the Condor central manager. Therefore, the combined usage of Condor-G and GlideIn allows aggregating the resources provided by many Globus HPCs in order to create a large and centralized cluster of resources coordinated by the Condor central manager. A major hurdle in the use of GlideIn jobs is the possibility that they may be prevented from contacting the central manager by the firewall of these clusters. For instance, GlideIn jobs communications were blocked by the firewall of the NGS clusters.

SWARM meta-scheduler has been designed to minimize the queuing time for PSA. It includes features necessary in a real Grid production environment such as failures management, dynamic and transparent reporting, and fair share usage policies. In order to offer both a user-friendly web-based environment and fast analyses, responsiveness has been considered as paramount in the design of the scheduler to be used in the GridQTL portal. The use of call-backs has therefore been preferred to the use of scheduling events, which power meta-schedulers such as AppLes and Nimrod/G. SWARM can speed up analyzes by a large factor on a Grid composed of HPCs belonging to the UK National Grid Service and local resources. Unlike GlideIn, SWARM does not need firewall settings to be adjusted as long as the HPCs used in the calculation allow outbound HTTP or HTTPS connections. Unlike AppLes, SWARM has not been designed to handle efficiently the scheduling of PSA with large input or output datasets. SWARM has not either been designed to take into consideration the economical cost of the resources usage on a world wide Grid as Nimrod/G does.

The SWARM meta-scheduler has been compared to an implementation of the MSR scheme and the gLite Resource Brokers. Comparisons show that the queuing time is lower for SWARM than for the other schedulers. Consequently LDLA analyses are completed significantly faster with SWARM than with the other methods. SWARM has been deployed within the LDLA beta-testing portal (GridQTL project), speeding up significantly computationally intensive analyses.

The SWARM source code will be distributed under the GNU lesser general public license and can be obtained from the authors or online on the GridQTL website [27]. Minimizing the queuing time has been shown to help speed up the execution of a large number of jobs. When the job duration is short, the queuing time is a critical factor. Some extensions could be brought to SWARM for other types of problems where the queuing duration can be long. For instance, it could also be extended so that MPI programs requiring a large number of CPUs may have to queue less until the right number of CPUs is available by scheduling them simultaneously on several clusters through a similar method.

V. ACKNOWLEDGMENTS

This work has made use of the resources provided by the Edinburgh Compute and Data Facility (ECDF). (http://www.ecdf.ed.ac.uk). The ECDF is partially supported by the eDIKT initiative (http://www.edikt.org). The authors would like to acknowledge the use of the UK National Grid Service in carrying out this work.

REFERENCES

[1] I. Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure," 2nd ed., Ed. Los Altos: Morgan-Kaufman, 2004.
[2] I. Foster, "Globus toolkit version 4: software for service-oriented systems," in *Proc. Conf. on Network and Parallel Computing*, Beijing, China, Nov.-Dec. 2005, pp. 2–13.
[3] J. Novotny, M. Russel and O. Wehren, "GridSphere: a portal framework for building collaborations," *Concurrency and Computation: Practice and Experience*, vol. 16, no. 5, pp. 503–513, Mar. 2004.
[4] G. Seaton, J. Hernández-Sánchez, J.-A. Grunchec, I. White, J. Allen, D.-J. De Koning, W. Wei, D. Berry, C. Haley and S. Knott, "GridQTL: A Grid Portal for QTL Mapping of Compute Intensive Datasets," in *Proc. 8th World Congress on Genetics Applied to Livestock Production*, Belo Horizonte, Brazil, Aug. 2006.
[5] M. Lynch and J. Walsh, "Genetics and Analysis of Quantitative Traits," Sunderland, MA: Sinauer Associates, 1998.
[6] T. Meuwissen, A. Karlsen, S. Lien, I. Olsaker and M. Goddard, "Fine mapping of a quantitative trait locus for twinning rate using combined linkage and linkage disequilibrium mapping," *J. Genetics*, vol. 161, no. 1, pp. 373–379, May 2002.
[7] The UK National Grid Service [Online]. Available: http://www.grid-support.ac.uk
[8] The Edinburgh Compute and Data Facility [Online]. Available: http://www.ecdf.ed.ac.uk/index.shtml
[9] The Condor Project [Online]. http://www.cs.wisc.edu/condor
[10] V. Subramani, R. Kettimuthu, S. Srinivasan and P. Sadayappan, "Distributed job scheduling on computational grids using multiple simultaneous requests," in *Proc. 11th IEEE Int. Symposium on High Performance Distributed Computing*, Edinburgh, UK, Jul. 2002, pp. 359–368.
[11] The Java Servlet Technology [Online]. Available: http://java.sun.com/products/servlet/index.jsp
[12] The NGS gLite Resource Broker tutorial [Online]. Available: http://wiki.ngs.ac.uk/index.php?title=Resource_Broker_Tutorial
[13] E. Laure, E. Fisher, S. Fisher, A. Frohner, C. Grandi and P. Kunszt, "Programming the Grid with gLite," *Computational Methods in Science and Technology*, vol. 12, no. 1, pp. 33–45, 2006.
[14] G. Gagliardi, "The EGEE European Grid infrastructure project," in *Proc. 6th Int. Conf. High Performance Computing for Computational Science*, Valencia, Spain, Jun. 2004, pp. 194–203.
[15] Job sumission into the LHC Grid (Job Management + JDL) [Online]. Available: http://www.egee.hu/grid06/download/day_1/05_EGEE_job_execution_and_JDL.ppt
[16] The LDLA beta testing portal [Online]. Available: http://cleopatra.cap.ed.ac.uk/gridsphere/gridsphere
[17] Apache Tomcat [Online]. Available: http://tomcat.apache.org
[18] J. Garret, "Ajax: A new approach to web applications", Adaptive path, 2005 [Online]. Available: http://www.adaptivepath.com/publications/essays/archives/000385.php
[19] R. Buyya, D. Abramson and J. Giddy, "Nimrod/G: An architecture for a resource management and scheduling system in a global computational Grid," in *Proc. 4th Int. Conf. on High Performance Computing in Asia-Pacific Region*, Beijing, China, May 2000, pp. 283–289.
[20] F. Casanova, G. Obertelli, F. Berman and R. Wolski, "The AppLeS parameter sweep template: user-level middleware for the Grid," in *Proc. Super Computing 2000*, Dallas, Texas, Nov. 2000.
[21] D. Abramson, J. Giddy and L. Kotler, "High performance parametric modeling with Nimrod/G: Killer application for the global Grid?," in *Proc. 14th Int. Parallel and Distributed Processing Symposium*, Cancun, Mexico, May 2000, pp. 520–528.
[22] S. Venugopal, R. Buyya and L. Winton, "A grid service broker for scheduling distributed data-oriented applications on global Grids," in

Proc. 2nd Int. Workshop on Middleware for Grid computing, Toronto, Canada, Oct. 2004, pp. 75-80.

[23] D. Abramson, R. Buyya and J. Gidd, "A computational economy for grid computing and its implementation in the Nimrod-G resource broker," *Future Generation Computer Systems*, vol. 18, no. 8, pp. 1061–1074, Oct.2002.

[24] B. Beeson, S. Melnikoff, S. Venugopal and D. Barnes, "A portal for grid-enabled physics," in *Proc. 2005 Australasian workshop on Grid computing and e-research - volume 44*, Newcastle, Australia, Jan.-Feb. 2005, pp. 13–20.

[25] T. Suzumara, H. Nakada, S. Matsuoka and H. Casanova, "GridSpeed: a Web-based Grid portal generation server," in *Proc. 7th Int. Conf. on High Performance Computing and Grid in Asia Pacific Region*, Tokyo, Japan, Jul. 2004, pp. 26–33.

[26] J. Frey, T. Tannenbaum, I. Foster and S. Tuecke, "Condor-G: a computation management agent for multi-institutional grids," *Cluster Computing*, vol. 5, no. 3, pp. 237–246, 2004, Jul. 2002.

[27] The GridQTL portal [Online]. Available: http://www.gridqtl.org.uk