Scheduling for Responsive Grids

Germain-Renaud, C LRI - CNRS and Universit Paris-Sud *et al*

20 December 2008

Journal of Grid Computing

EGEE-II is a project funded by the European Commission
Contract number INFSO-RI-031688

Scheduling for Responsive Grids

Cécile Germain-Renaud
LRI and LAL

Charles Loomis
LAL

Jakub T. Mościcki
CERN

Romain Texier
LRI

June 2006

Abstract. Grids are facing the challenge of seamless integration of the grid power into everyday use. One critical component for this integration is responsiveness, the capacity to support on-demand computing and interactivity. Grid scheduling is involved at two levels in order to provide responsiveness: the policy level and the implementation level. The main contributions of this paper are as follows. First, we present a detailed analysis of the performance of the EGEE grid with respect to responsiveness. Second, we examine two user-level schedulers located between the general scheduling layer and the application layer. These are the DIANE (DIstributed ANalysis Environment) framework, a general-purpose overlay system, and a specialized, embedded scheduler for gPTM3D, an interactive medical image analysis application. Finally, we define and demonstrate a virtualization scheme, which achieves guaranteed turnaround time, schedulability analysis, and provides the basis for differentiated services. Both methods target a brokering-based system organized as a federation of batch-scheduled clusters, and an EGEE implementation is described.

Keywords: Responsiveness, Interactive Grids, Meta-scheduler, User-level Scheduling

1. Introduction

The exponential increases in network performance and storage capacity [41], together with ambitious national and international efforts, have already enabled the virtualization and pooling of processors and storage in advanced and relatively stable systems such as the EGEE grid. However, it is more and more evident that the exploitation model for these grids is somehow lagging behind. At a time where industry acknowledges interactivity as a critical requirement for enlarging the scope of high performance computing [35, 43, 6], grids cannot anymore be envisioned only as very large computing centres providing batch-oriented access to complex scientific applications with high job throughput as the primary performance metric.

A much larger range of grid usage scenarios is possible. Seamless integration of the grid power into everyday use calls for unplanned and interactive access to grid resources. We define *responsive grids* as grid infrastructures that support on-demand computing and interaction. This paper describes a set of scheduling methods providing different levels and types of Quality of Service (QoS) required by responsiveness.

Compared to many recent proposals in this area, our methods target production grids. They have been implemented within EGEE, on top of the gLite middleware. EGEE is the largest production grid worldwide, comprising more than 20000 CPUs, 200 sites and 20000 jobs per day and requiring the strongest constraints on dependability. In this framework, responsiveness must be built on top of the traditional grid scheduling tools, which are batch-oriented and dominated by fair-share policies at institutional time-scales. The associated constraints are:

- delays incurred by non-interactive jobs are bounded,
- resource utilization is not degraded (e.g. by idling processors), and
- the local policies governing resource sharing (Virtual Organizations, advance reservation, etc.) are not impacted.

This rest of this paper is organized as follows. Section 2 describes use-cases for grid responsiveness. Section 3 presents the scheduling architecture of the EGEE grid and an experimental study of the EGEE profiles of execution time and overhead. Section 4 presents two examples of user-level scheduling deployed on top of gLite, which is the EGEE middleware. The first one exemplifies a generic overlay system. The second one is an application-dedicated environment, which exemplifies grid-enabled computational steering in medical image analysis. We show that user-level scheduling does improve the quality of service, by eliminating the middleware overhead, providing a sustained job output rate, and optimizing the failure recovery. On the other hand, user-level scheduling is limited to best-effort. Section 5 presents the Virtual Reservation scheme, which provides guarantees on the overall turnaround time, and its implementation inside gLite. Section 5 discusses related work, and Section 6 presents the conclusions.

2. Motivation

Responsiveness is a key component for real-world grid usage; this section presents a few examples. The first one is grid-enabling medical image analysis [11, 45, 23]. In a clinical context, medical image analysis (segmentation, registration) and exploitation (augmented reality for intervention planning or intra-operative support) require full interaction because computer programs are not yet competitive with the human visual system for mining these structured and noisy data. Analyzing large images at a sufficient speed to support smooth visualization requires not only substantial computing power, which can be provided by the grid, but also unplanned access and sophisticated interaction protocols.

The second use case is digital libraries. Most of the resource consumption in digital libraries management is related to bulk, off-line tasks such as indexing. When humans query this massive amount of data, various actions are triggered such as feature extraction in a query-by-example scheme, which must take place before the actual search can be carried out, or content protection (e.g. watermarking). User satisfaction requires nearly instantaneous response.

Finally, in the larger perspective of ubiquitous computing and ambient intelligence, multi-modal interfaces that are capable of natural and seamless interaction with and among individual human users are mandatory. Responsiveness is a key component for grid-enabling the methods and technologies that form the back-end of these interfaces, such as pattern analysis, statistical modelling and computational learning.

Interactive grid applications require a specific grid guarantee, namely a bound on the overall turnaround time of the grid jobs contributing to the application. Because such jobs have typically a short execution time and require completion by a deadline, we call them *Short Deadline Jobs* (SDJ) in the remainder of this paper.

3. A case for responsiveness

3.1. EGEE Scheduling

EGEE combines globally-distributed computational and storage resources into a single production infrastructure available to EGEE users. Each participating site configures, runs, and maintains a batch system containing its computational resources and makes those resources available to the grid via a gatekeeper. The scheduling policy for each site is defined by the site administrator. Common scheduling policies use either FIFO (often with per-user or per-group limits) or fair-share algorithms. Consequently the overall EGEE scheduling policy is not centrally defined, but the effect of the interaction of largely autonomous policies.

The gLite middleware deployed on the EGEE infrastructure integrates the sites' computing resources through the Workload Management System (WMS) [3]. The WMS is a set of middleware-level services responsible for the distribution and management of jobs. The site computational resources present a common interface to the WMS, the Computing Element (CE) service. The CE specification is one of the core parts of the Glue information model [4], which is the current basis for interoperability between EGEE and other grids. From the middleware point of view, a CE has multiple functions: running jobs, staging the files required by the job, providing information about resource availability, and notifying the WMS of the job-related events. In the framework of this paper, a CE can be simply considered as a batch queue, subject to the above-mentioned policies.

The core of the WMS is the Workload Manager which accepts jobs from users and dispatches them to computational resources based on the users requirements on one hand, and the characteristics (*e.g.* hardware, software, localization) and state of the resources on the other hand. The WM is implemented as a distributed set of resource brokers, with some tens of them currently installed; all the brokers get an approximatively consistent view of the resource availability through the grid information system. Each broker reaches a decision of which resource should be used by a matchmaking process between submission requests and available resources. Job requirements are exposed to the various services of the WMS via the Job Description Language (JDL) [38], derived from the Condor ClassAd language [39]. The users can rank acceptable resources (in JDL language) by using an arbitrary expression which uses state information published by the resources. Once a job is dispatched, the broker only reschedules it if it failed; it does not reschedule jobs based on the changing state of the resources.

3.2. EGEE USAGE

The relevant quantities for measuring the responsiveness of the grid are the running time t, the on-site queuing delay q, and the middleware overhead s, which includes the various delays experienced by the job in the WMS. The turnaround time $m = s + q + t$ is the total time from submission to notification that the job has completed. For the study presented here, these quantities were derived from information in the

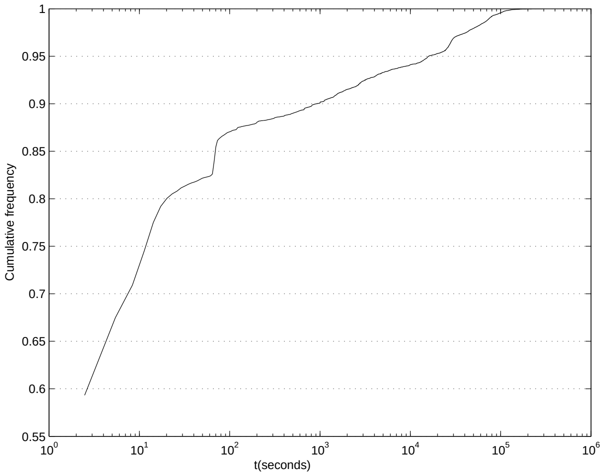

Figure 1. Cumulative distribution of execution times.

Logging and Bookkeeping service (LB). This is a companion service to the resource broker which maintains the state of all jobs managed by the resource broker.

Because the detailed LB data were not available for all jobs, the analysis below is limited to a particular broker (grid09.lal.in2p3.fr). These data cover one year (October 2004 to October 2005) and include more than 50000 successful production jobs from 66 distinct users. Fig. 1 shows the distribution of execution time from this trace. The striking feature is the importance of short jobs: the 80% quantile is 20 s. The second important point is the dispersion of t; the mean is 2 s, but the standard deviation is of the order of 10^4 s. The very large fraction of extremely short jobs is partially due to the high usage of this particular broker by the EGEE Biomed Virtual Organization. However, for more than 50% of the overall EGEE jobs at the same period, the execution time is less than 3 minutes.

Fig. 2 shows the distribution of the dimensionless *overhead factor* $o_r = (m - t)/t$, which is the overhead normalized by the execution time. The leftmost histogram shows the distribution of the full sample: only 26% of the 53000 jobs are in the first bin, meaning than 74% of the jobs suffer an overhead factor larger than 25. A closer look at the small overheads (rightmost histogram) shows that only 13% of the jobs experience an overhead factor lower than 2. Clearly, the EGEE infrastructure can make no claims for responsiveness using only the base middleware services.

The next question is the respective impacts of the middleware and the queuing time on the global overhead. Fig. 3 plots the distribution

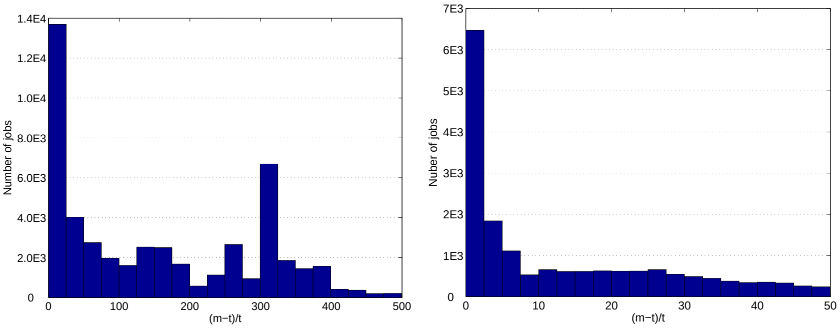

Figure 2. Distribution of the overhead factor.The left histogram is the distribution of the full sample, the right histogram is the distribution of the small overheads.

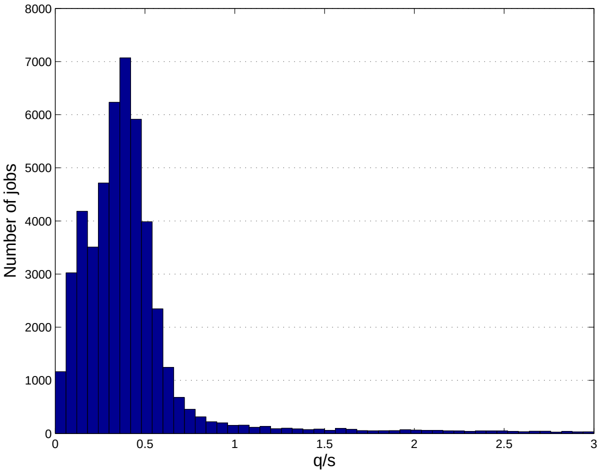

Figure 3. Impact of the queuing time and the middleware on the overhead.

of q/s, and shows that the queuing time is a significant component of the overhead. This behaviour was exhibited at an early stage of EGEE usage, where the pressure on the resource was only starting to increase. Finally, the median queuing time is 91 seconds, and the median middleware overhead is 221 seconds.

4. User-level scheduling

Submitting, scheduling and mapping of jobs on a grid take at least one order of magnitude more time than the execution time for SDJ even in absence of competition for resources. For instance, with the most recent and tuned EGEE middleware, gLite 3.0, the middleware latency remains on the order of minutes. User-level scheduling is the most promising way to address the difference of scale between short execution times and the large grid middleware latencies.

User-level (or application-level) scheduling is a virtualization layer on the application side. Instead of being executed directly, the application is executed via an overlay scheduling layer (user-level scheduler). The overlay scheduling layer runs as a set of regular user jobs and therefore it operates entirely inside user space. Because user-level scheduling does not require any modification to the Grid middleware and infrastructure nor the deployment of special services in the Grid sites, it provide immediate exploitation of the full range of a Grid sites which are available for a given user.

The user-level scheduling approach has the following constraints:

- user jobs must be instrumented with the scheduling functionality, and
- jobs run under user-level scheduling must compete on the same basis with all other jobs on the grid, and their resource usage be fully reported to the corresponding user.

A user-level scheduler may be embedded into the application or external to it. A scheduler embedded into the application is developed and optimized specifically for a given application, typically by re-factoring and instrumenting the original application code. It allows fine tuning and customizing the scheduling according to the specific execution patterns of the application. Such a scheduler is intrusive at the application source code level which means that the code reuse of the scheduler is reduced and the development effort is high for each application. A scheduler external to the application relies on the general properties of the application such as a particular parallel decomposition pattern (e.g. iterative decomposition, geometric decomposition or divide-and-conquer). An application adapter connects the external scheduler to the application at runtime. Depending on the decomposition pattern, the application re-factoring at the source code level may or may not be required. The disadvantage of external schedulers is that it may be very hard to generalize execution patterns for irregular or speculative parallelism. In this case, which occurs in various situations ranging from medical image processing to portfolio optimization [50], a development of a specialized embedded scheduler may be necessary.

In the next sections we examine two user-level schedulers: an external scheduler for generic master-worker applications (DIANE) and an embedded scheduler for medical image processing (gPTM3D).

4.1. DIANE: a generic, external scheduler

4.1.1. *Overview*

DIANE (DIstributed ANalysis Environment) is a R&D project developed in the Information Technology Department at CERN in Geneva Switzerland. [36]. It is a generic user-level scheduler based on the extended task farm (master/slave) processing . The runtime behaviour of the framework, such as failure recovery or task dispatching, may be customized with a set of hot-pluggable policy functions. This enables fine-tuning of the scheduler according to the needs of particular application and provides support for other parallel decomposition patterns (e.g. divide-and-conquer).

4.1.2. *Applications*

DIANE provides a python-based framework and enables a rapid integration with existing applications. Both transparent and intrusive application integrations have been demonstrated. Data analysis in Athena framework for Atlas experiment [1], is an example of transparent application integration; the application adapters in the form of python packages have been developed without modifying the original application code. The examples of intrusive integrations include the particle simulation in medical physics using Geant 4 toolkit [22]. The parallelization of these applications has been based on the iterative decomposition and master/worker processing model with fully independent tasks. Other applications using DIANE include, among others, the Geant 4 statistical regression testing application [34], Autodock [10] tools for bioinformatics and telecommunication applications [32].

4.1.3. *Execution model*

In the DIANE execution model, a temporary virtual master/worker overlay network is created for each user job and is destroyed when the job terminates. The job is split into a number of tasks which are executed by a number of lightweight worker agents in the Grid. The worker agents run as regular Grid jobs submitted with credentials and authentity of a single user. Therefore the full user-based accounting from the system administration point of view is possible. The agents are time-limited and the computing resources are released when the processing terminates (all tasks processed) or if they exceed the time limit on the batch queue, whatever occurs first. The number of resouces acquired by a user is limited by standard mechanisms i.e. the fair-share policies in the Grid and in the local sites.

Each task is defined by a set of application-specific parameters. The dispatching of tasks is the process of allocating the tasks to workers by sending appropriate parameters to the worker agents. The communication overhead is typically much smaller than in the systems based on checkpointing and task migration and it allows scheduling with a high rate of incoming and outgoing tasks. For example the DIANE Master routinely achieves peaks of 110-120 Hz without observable degradation in the performance. This means that scheduling overhead is negligible for $N * 120$ worker agents if average task duration is N seconds.

The default scheduling algorithm used in DIANE is based on dynamic *pull* approach also known as *self-load-balancing*. DIANE makes it easy to plug-in alternative algorithms, however the results described in this paper use the default one.

DIANE allows the standard GSI-based authentication and authorization of the worker agents. The Grid proxy certificate is shipped via standard Grid submission mechanisms to the worker node, while the master retains the original. The secure mode prevents the accidental mixing of user credentials in a single overlay. The results described in this paper refer to the default, non-authenticated mode.

The following sections present three examples of improved QoS characteristics with DIANE User Level Scheduling: the job turnaround time, job completion rate, and error recovery.

4.1.4. *Job turnaround time with high-granularity splitting*

DIANE supports high-granularity job splitting, i.e. partitioning a job into a large number of short or very short tasks. For example, the radio-frequency compatibility analysis jobs for ITU RRC06 conference [32], have been split into approximately 40 000 tasks performed simultaneously by around 200 worker agents at 6 EGEE Grid sites across Europe.

Task duration was highly variable (Fig. 4) lasting from few seconds (majority of the tasks) to 20 minutes (few individual tasks). The exact distribution of the task duration was not known until the job was fully executed. Consequently, it was not possible to *a priori* aggregate short tasks and isolate long tasks. The efficiency of user-level scheduling was high with the number of tasks executing in parallel very close to the size of the worker pool (Fig. 5). As shown in previous sections (Fig. 2) the job turnaround time is orders of magnitude higher in a plain grid environment.

4.1.5. *Job completion rate*

User-level scheduling provides a more sustained job completion rate. Fig. 6 shows the job completion rate for a Geant 4 release statistical regression testing application [34]. The job has been split in 207 tasks and average task duration was around 400 seconds. In the Grid, the

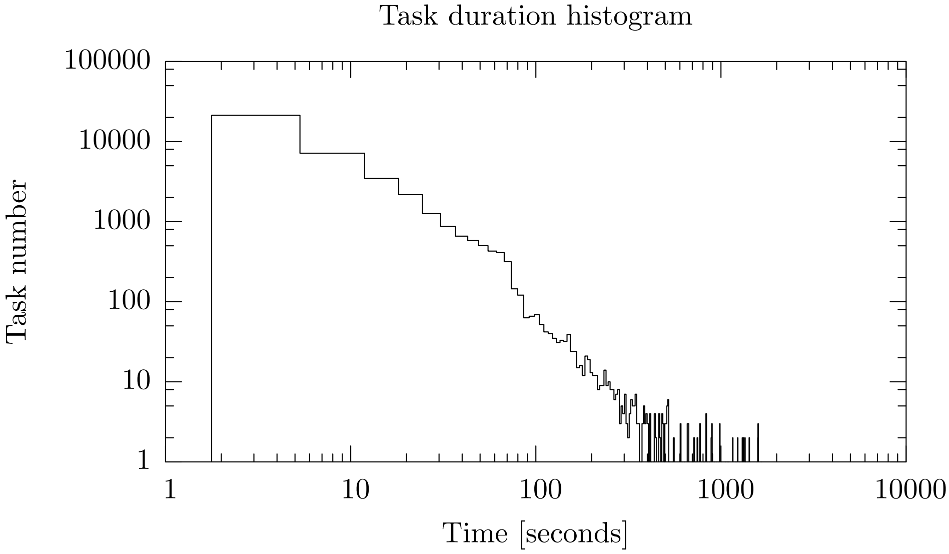

Figure 4. High-granularity splitting with exponential distribution of the task execution time. Most of 40000 tasks execute in less then 10 seconds, with individual tasks executing in 1000 seconds.

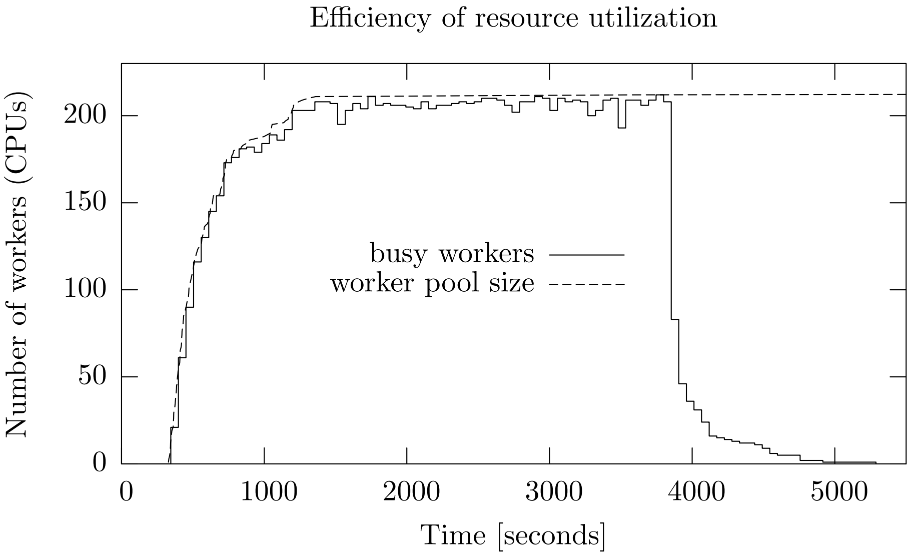

Figure 5. Comparison of the number of concurrently processed tasks (the number of busy workers) and the number of available workers (the worker pool size). The difference represents the scheduling overhead, including the network communication cost. Currently, the scheduler does not remove excessive workers from the pool, hence the number of idle workers increases at 4000s due to few long-lasting tasks.

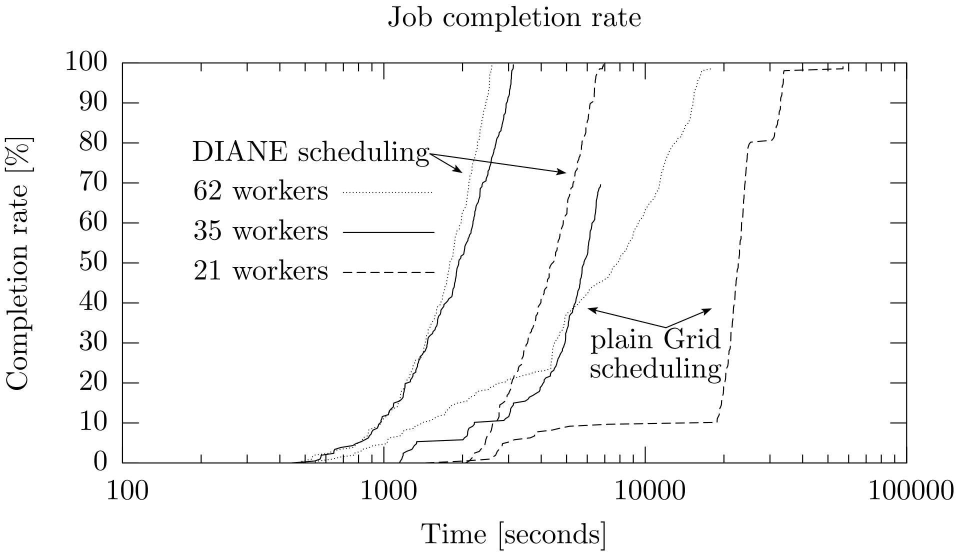

Figure 6. Comparison of job completion rate between user-level scheduling based on DIANE (A) and plain Grid scheduling(B). Geant 4 regression testing jobs were run simultaneously in both scheduling modes. Equal number of available computing resources (85 worker nodes) within EGEE Grid in each mode was guaranteed. The figure shows three selected jobs with typical behaviour. This figure has been taken from [34].

load on the Computing Elements (queuing time) and the load on the Resource Broker (efficiency of matchmaking) may change dynamically in short periods of time resulting in a job completion curve which is less predictable (B_1 and B_3) or jobs being stuck in the Grid for a very long time and appear as incomplete (B_2). The user-level scheduler assures that, even if the number of effectively available resources is low and varying, the job output throughput is stable if splitting granularity is correctly chosen.

4.1.6. *Error recovery*

Efficient and accurate failure recovery is an important factor for Quality of Service. Large distributed systems such as the grid are prone to diverse configuration and system errors. A generic strategy of handling errors does not exist and the specific strategies depend on the application as well as the environment. An application-oriented scheduler such as DIANE is capable of distinguishing application and system errors and reacting appropriately via customizable error recovery methods. Crashing worker agents are automatically taken out of the worker pool. Transient connectivity problems in the WAN are detected; the failed tasks are automatically re-dispatched to another worker agents. The mechanism uses a direct, highly efficient communication links in the virtual master/worker network and is much more efficient than a standard metascheduling techniques implemented in the middleware (JDL RetryCount parameter) which involve the full submission cycle.

A part of recent Avian Flu Drug Search [29] have been performed using DIANE scheduler. A master agent spanning several weeks was taking care of efficient error recovery so the system could be operated by a single person. Because of the long duration of the job, the worker agents were often aborted because they exceeded the time limits in the queues at the Computing Elements. The operator was adding new worker agents to the system so that at least 200 were available at any time. DIANE was able to dynamically reconfigure the virtual master/worker network to accommodate the new worker agents. The overall efficiency of DIANE user-level scheduling was 84%, compared to 38.4% efficiency of pure grid scheduling.

4.2. gPTM3D

PTM3D [42] is a fully-featured DICOM image analyzer developed at LIMSI. PTM3D transfers, archives and visualizes DICOM-encoded data. Besides moving independently along the usual three axes, the user is able to view the cross-section of the DICOM image along an arbitrary plane and to move it. PTM3D provides computer-aided generation of three-dimensional (3D) representations from CT, MRI, PET-scan, or echography 3D data. A reconstructed volume (organ, tumour) is displayed inside the 3D view. The reconstruction also provides the volume measurement required for therapeutic decisions. The system currently runs on standard PC computers and it is used online in radiology centres. Clinical motivation for grid-enabled volume reconstruction is described in [21].

The first step in grid-enabling PTM3D (gPTM3D) is to speedup compute-intensive tasks such as the volume reconstruction of the whole body used in percutaneous nephrolithotomy planning [37]. The volume reconstruction algorithm includes a semi-automatic segmentation component based on an active contours method where the user initiates the segmentation, and can correct it at anytime. It also includes a tessellation component which is the compute-intensive part of the algorithm. The gPTM3D application requires fine-grained parallelism. The parallel tasks are the reconstruction of one slice; in the examples presented Fig. 7, the execution time of the majority of the tasks is in the order of a few hundreds of milliseconds but with high variability.

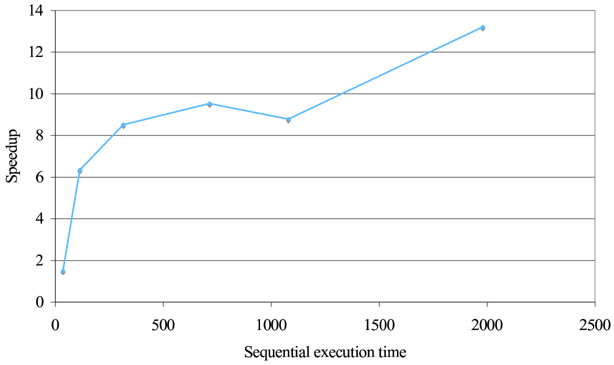

Figure 7. gPTM3D performance

When the geometry of the volume becomes complex, the reconstruction of the critical slices can last for 20 seconds or more.

The architecture has two components: scheduler/worker agents at the user-level and the Interaction Bridge (IB) as an external service. The IB acts as a proxy between the PTM3D workstation, which is not EGEE-enabled and the EGEE world. When opening an interactive session, the PTM3D workstation connects to the IB. In turn, the IB launches a scheduler and a set of workers on an EGEE site, through fully standard requests to an EGEE User Interface. A stream is established between the scheduler and the PTM3D front-end through the IB. When the actual volume reconstruction is required, the scheduler receives contours. The scheduler/worker agents follow a pull model with each worker computing one slice of the reconstructed volume at a time, and sending it back to the scheduler, which forwards them to IB from where they finally reach the front-end.

The overall response time is compatible with user requirements (less than 2 minutes), while the sequential time on a 3GHz PC with 2GB of memory can reach 20 minutes and more than 30 minutes on less powerful front-ends. So far, the only bottleneck is the rate at which the front-end is able to generate contours. Fig. 7 presents the speedup achieved on EGEE, with one scheduler and up to 14 workers in the largest case. For small reconstructions, the grid is obviously not necessary; we have included them to prove that there is no penalty (in fact a small advantage) in this case. Thus there is no need to switch from a local mode to a grid one in an interactive session. For the largest reconstruction, the speedup is nearly optimal. Lowering the execution time to this point has strictly no impact on the local interaction scheme, which includes stopping, restarting and improving locally the segmentation.

5. Grid differentiated services

5.1. Virtual Reservations

As a shared resource, a grid supports a broad spectrum of workloads ranging from long-running batch workloads executed under best-effort policy to workflows [28, 20] or parallel applications for which specific scheduling strategies have been proposed. Examples of these strategies include static [18] or dynamic [47] gang-scheduling using advance reservation [38] and middleware mechanisms favouring simultaneous allocation such as the EGEE DAG job type. Grid advance reservation suffer from two drawbacks: first, planning is not consistent with the goal of seamless integration with everyday computing practice, for instance the use cases described in Section 2; second, reservation is inherently not work-conserving, meaning that processors might idle while eligible jobs are queued [46].

Providing differentiated QoS either at the processor or network level usually relies on some implementation of Generalized Processor Sharing (GPS). However, the fundamental concept required for schedulability analysis and schedule construction in these frameworks is that the allocation of resources may be broken along quanta of time. The problem for grid scheduling is that such quanta do not exist. Jobs are not partitionable. Except for checkpointable jobs, a job that has started running cannot be suspended and restarted later. Moreover, as shown before, the execution times exhibit an extremely high variance.

We have defined and implemented the concept of a *Virtual Reservation* (VRes), which addresses both issues of advance reservation and scheduling quanta by allowing controlled time-sharing. VRes permits the definition of time quanta and their exposure at the grid level.

At the site level, each of the p physical processors is virtualized into k virtual processors, providing pk slots to the site scheduler. When a virtual slot is unused, the computing bandwidth is transparently returned to the other classes sharing the same physical processor. Thus, a fraction of these slots can then be permanently reserved for some class of applications without jeopardizing utilization.

The mapping of classes first to the virtual processors, then onto the physical ones is obviously the key for full processor utilization. This mapping must be controlled so that each class maps to the full range of physical processors, as shown in Fig. 8. Provided that the mapping is controlled, the reservation ensures both application isolation with respect to computational bandwidth and full processor utilization.

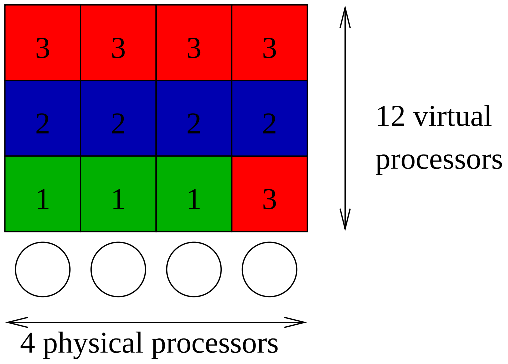

Figure 8. Example of VRes: class 1, 2 and 3 are allocated respectively 1/4, 1/3 and 5/12 of the computational bandwidth

5.2. EGEE Implementation

An implementation of VRes has been developed for the MAUI scheduler and the gLite middleware. It can be downloaded from the EGEE SDJ Working Group site http://egee-na4.ct.infn.it/wiki/index.php/ShortJobs. The Job Description language (JDL) has been modified to include a Boolean attribute SDJ. Sites willing to accept SDJ jobs set up a CE which permits running one job per SDJ slot. Jobs submitted to this CE either are immediately scheduled or rejected. The broker is notified in case of rejection and can either reschedule the job on another resource or notify the user. These sites also configure their scheduler with parameters controlling the computational bandwidth dedicated to SDJ. In particular, the wall-clock time and CPU time of SDJ jobs are limited. While these parameters are lower for SDJ jobs than for the usual batch jobs, all EGEE jobs are subject to the same kind of limitations, and all are aborted if they exceed these.

This work has exposed a problem with scheduling in the EGEE middleware. The system does not permit a CE to provide access control based on job type, which is required for application isolation in general and for QoS in our case. As a temporary solution, a name-based dispatch has been set up in gLite 3.2. The SDJ-dedicated CEs

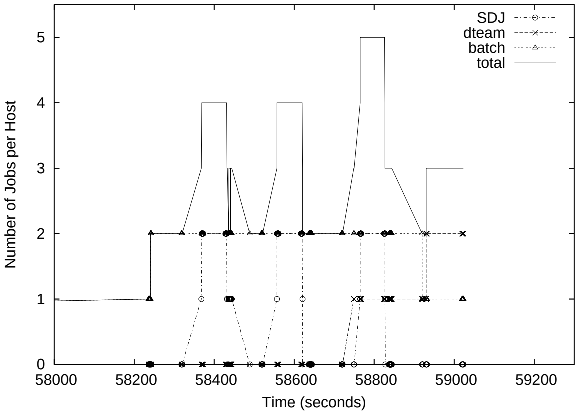

Figure 9. Number of concurrent jobs on a single dual-processor node as a function of time.

are named such that they have a trailing "sdj". The submission system introduces an appropriate regular expression in job requirements so as the WMS will select select SDJ CEs for short deadline jobs and prevents batch jobs from being scheduled on SDJ CEs. It is worth mentioning that this method can be adopted for early experiments of other classes, because it requires only minor modifications of the gLite code. A more elegant and general solution is being investigated. However, the Glue schema must be modified and such modifications are a long process.

Tests that have been conducted at LAL to ensure the correct behaviour of the SDJ configuration. Fig. 9 shows a breakdown of the occupation of one dual-processor node. On a background of batch jobs, which never exceed 2 (one per processor), SDJ can run within the same limit, and also concurrently with a third class (dteam) required by EGEE operational monitoring. Hence there are five slots per dual-processor node. Fig. 10 exemplifies control of the global computational bandwidth at the site level dedicated to SDJ. In this configuration, a maximum of ten concurrent SDJ were permitted.

The virtual reservation mechanism and the SDJ CE have been put in production at LAL since May 2006. The LAL site is equipped with a mixture of IBM and HP 1U rack-mounted dual-processor (AMD Opteron, 2.2 GHz) machines with 1 GB of RAM per CPU (2 GB total) and 80 GB of disk. The SDJ slots are routinely used in production by several biomedical applications and also for EGEE demonstrations (one cannot wait in queues when the audience is waiting for a live demonstration), and run concurrently with the usual batch jobs. The site utilization is extremely high, approaching a steady 100%. This experimental result provides an empirical answer to the often raised

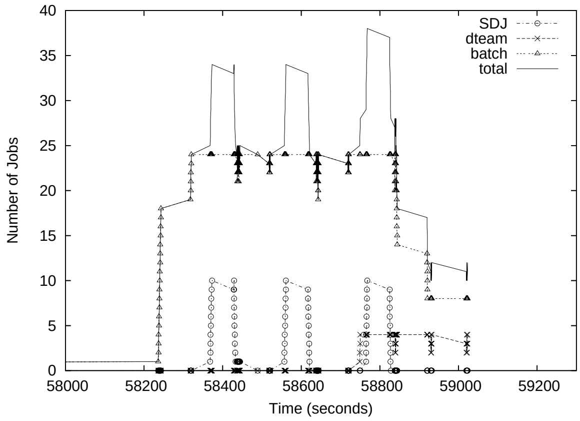

Figure 10. Number of concurrent jobs on the site as a function of time.

issue of the negative impact of concurrency (from cache to IO) on real-world workloads running on high-end processors.

6. Related work

Existing approaches to grid scheduling for QoS follow three distinct paths: Virtual Machines (VM) encapsulation, statistical prediction, and service level agreements. Virtual machines provide a powerful new layer of abstraction in centralized computing environments in order to ensure fault isolation. Distributed scheduling based on VM encapsulation has been explored as a general tool in the PlanetLab project [7]. The Virtuoso project has more specifically explored virtualization for differentiated services [30, 31], and the Virtual Workspaces project [27] investigates the large-scale deployment of VM inside the Globus middleware. Virtual machines provide complete freedom of scheduling and even migrating an entire OS and associated computations which considerably eases time-sharing between deadline-bound short jobs and long running batch jobs. On the other hand, the virtual machines strategy is extremely invasive. All of, or a significant fraction of, the computations must be run inside virtual machines to provide scheduling opportunities—something for which traditional batch users have little incentive. Another issue is that VM interactivity follows the remote desktop model. In this model, which has been often been adopted for grid-enabling computational steering [44, 24, 26, 40], the user front-end is a passive terminal. With Grid Differentiated Services and user-level scheduling, we provide a much more modular environment that can support any combination of local and remote computations.

Accurate statistical prediction of the workloads is possible in large range of situations including shared clusters [16] and batch-scheduled parallel machines [13]. In particular, [48] shows that statistical prediction allows efficient support of interactive computations in unreserved cluster environments. At the grid scale, in the current status where time-sharing is possible only through control mechanisms such as VRes, predictive methods would apply for instance to the availability of SDJ slots provided by VRes.

Service level agreements (SLAs) are the standard to represent the agreed constraints between service consumers and service providers on a grid [2]. SLAs by themselves do not provide scheduling solutions, but allow expressing flexible requirements and incorporating multi-criteria approaches. SLAs could be applied to differentiated services in our context. For instance proposing a choice between a quick and reliable turnaround time, with strong completion constraints, and a more unreliable turnaround time without constraints. SLAs also offer the perspective of a general framework for renegotiation of resources [33] by running jobs. In our context this could be used to switch from the first mode to the second one, for instance when a SDJ approaches the end of its allocated time and must be prorogated.

User-level scheduling has been proposed in many other contexts, and a case for it has been made in the AppLeS [14, 8] project. In a production grid framework, the DIRAC [51] project has proposed a permanent grid overlay where scheduling agents pull work from a central dispatching component. Our work differs from DIRAC on a major point: both for DIANE and gPTM3D, the execution agents are regular gLite jobs, and are thus subject to all grid policies and accounting. The abuse of glideIn techniques, which would permanently launch execution agents, would be counter-productive. The local EGEE schedulers (typically MAUI or PBS) do enforce fair share across VO and users. Thus, running a useless execution agent will prevent it to be run on the same site at the next scheduler decision. Obviously, if the site allows infinite execution, there will never be a scheduler decision, but the resource usage of this agent would be charged to the appropriate user or VO.

7. Conclusion

We have presented complementary strategies to address the QoS requirements of a responsive grid: Grid Differentiated Services and user-level schedulers. Grid Differentiated Services provide a general framework for the isolation of classes of applications and the realization at the grid level of the concepts required for hard or soft real-time scheduling. User-level schedulers cope with high latencies associated with grid middleware. Equally important is a clean separation between two optimization problems: at the grid level, the optimization is related to fair-share and load balancing, while at the user-level, the optimization is for a specific application workload. Depending on the application requirements, Grid Differentiated Services and user-level schedulers can be used separately or combined. In the example of gPTM3D, combining Grid Differentiated Services and an embedded user-level scheduler provides a fully transparent coupling of the grid resources with an augmented reality desktop software. The scope of applications deployed on top of the DIANE generic scheduler exemplify the impact of user-level scheduling on a number of QoS characteristics.

Both strategies have been deployed on the EGEE grid, as autonomous site decisions (for the Grid Differentiated Services) or as regular user jobs (for the user-level schedulers). They are fully compatible with gLite, the existing EGEE middleware. Their architecture and to a large extent their implementation depend only on generic grid concepts. We are convinced that this non-intrusiveness is a key to a progressive convergence of QoS and grid technology.

Acknowledgements

This work was partially funded by the EGEE EU project (INFSO-RI-508833 Grant). gPTM3D is part of the AGIR project funded by the ACI Masses de Données program of the French ministry of research .

References

1. Atlas Computing - Technical Design Report *CERN-LHCC-2005-022.* http://doc.cern.ch//archive/electronic/cern/preprints/lhcc/public/lhcc-2005-022.pdf
2. R. AlAli, K. Amin, G. von Laszewski, O. Rana, D. Walker, M. Hategan and N. Zaluzec. Analysis and Provision of QoS for Distributed Grid Applications. *Journal of Grid Computing*, 2(2):163-182. June 2004
3. P. Andreetto et al. Practical approaches to Grid workload and resource management in the EGEE grid. In *Procs. CHEP'04*
4. S. Andreozzi et al. GLUE Schema Specification version 1.2. http://infnforge.cnaf.infn.it/glueinfomodel/
5. S. Baruah, N. Cohen, C. Plaxton and D. Varvel. Proportionate Progress: A Notion of Fairness in Resource Allocation. *Algorithmica* 15(6), pp. 600-625, 1996.
6. S. Basu, V. Talwar, B. Agarwalla and R. Kuma. Interactive Grid Architecture for Application Service Providers. HP Technical Report HPL-2003-84R1. 2003.

7. A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, S. Muir, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak. Operating System Support for Planetary-Scale Network Services. In *Procs. of the First Symposium on Networked Systems Design and Implementation (NSDI*, pp 253–266. 2004.
8. F. Berman, R. Wolski and H. Casanova. Adaptive Computing on the Grid Using AppLeS. *IEEE Trans. Parallel Distrib. Syst.*, 14(4): 369–382. 2003.
9. G. Shao and R. Wolski and F. Berman: Master/Slave Computing on the Grid; in Proceedings of the 9th Heterogeneous Computing Workshop, Cancun, Mexico, May 2000, pp. 3-16. http://citeseer.ist.psu.edu/wolski00masterslave.html
10. Morris, G. M. et al.(1998), Automated Docking Using a Lamarckian Genetic Algorithm and and Empirical Binding Free Energy Function J. Computational Chemistry, 19: 1639-1662
11. D. Berry, C. Germain-Renaud, D. Hill, S. Pieper and J. Saltz. Report on the Workshop IMAGE'03: Images, medical analysis and grid environments. TR UKeS-2004-02, UK National e-Science Centre, Feb. 2004.
12. Christophe Blanchet et al. GPSA: Bioinformatics grid portal for protein sequence analysis on EGEE grid. In *Procs of Healthgrid 2006*, Studies in Health Technology and Informatics 120, IOS Press. May 2006
13. J. Brevik, D. Nurmi and R. Wolski. Predicting bounds on queuing delay for batch-scheduled parallel machines. In *Procs. of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP '06)*, pp 110–118. 2006.
14. H. Casanova, G. Obertelli, F. Berman and R. Wolski. The AppLeS parameter sweep template: user-level middleware for the grid. In *Procs 2000 ACM/IEEE conference on Supercomputing SC'00.* 2000.
15. A. Chandra, M. Adler, and P. Shenoy. Deadline fair scheduling: Bridging the theory and practice of proportionate-fair scheduling in multiprocessor servers. In *Procs. of the 7th IEEE Real-Time Technology and Applications Symposium.* June 2001.
16. P. Dinda. Design, Implementation, and Performance of an Extensible Toolkit for Resource Prediction In Distributed Systems. *IEEE Transactions on Parallel and Distributed Systems*, 17(2). 2006.
17. D.H.J. Epema, M. Livny, R. van Dantzig, X. Evers and J. Pruyne. A worldwide flock of Condors: Load sharing among workstation clusters. *Future Generation Computer Systems*, 12:53–65. 1996.
18. D. G. Feitelson and M. Jette. Improved Utilization and Responsiveness with Gang Scheduling. In *Procs of the IPPS '97 Workshop on Job Scheduling Strategies for Parallel Processing*, 1997
19. I. Foster, M. Fidler, A. Roy, V, Sander and L. Winkler. End-to-End Quality of Service for High-end Applications. *Computer Communications*, 27(14):1375-1388. 2004.
20. T. Glatard, J. Montagnat and X. Pennec. Efficient services composition for grid-enabled data-intensive applications. In *Procs of the IEEE Int. Symp. on High Performance Distributed Computing (HPDC'06)*, June 2006.
21. C. Germain-Renaud, R. Texier and A. Osorio. Interactive Reconstruction and Measurement on the Grid. *Methods of Information in Medecine*, 44(2):227- 232. 2005.
22. S. Guatelli et al.: Geant4 Simulation in A Distributed Computing Environment; *submitted to IEEE Trans. Nucl. Sci. 2006*
23. S. Hastings, M. Kurc, S. Langella, U. V. Catalyurek, T. C Pan and J. H. Saltz, Image Processing for the Grid: A Toolkit for Building Grid-enabled Image

Processing Applications. In *Proc. 3rd Int. Symp. on Cluster Computing and the Grid*, pp 36–43. 2003

24. P. Heinzlreiter and D. Kranzlm??ller. Visualization Services on the Grid: The Grid Visualization Kernel. *Parallel Processing Letters* 13(2):135–148. 2003.
25. E. Heymann, M. A. Senar and E. Luque and M. Livny". Adaptive Scheduling for Master-Worker Applications on the Computational Grid. In *Procs 1st IEEE/ACM Intl Workshop on Grid Computing (GRID 2000).* 2000
26. R. Kumar, V. Talwar and S. Basu. A resource management framework for interactive Grids. *Concurrency and Computation: Practice and Experience*, 16(5):489–50. April 2004.
27. K. Keahey, I. Foster, T. Freeman, and X. Zhang. Virtual Workspaces: Achieving Quality of Service and Quality of Life in the Grid. To appear in *The Scientific Programming Journal*. 2006.
28. G. von Laszewski, M. Hategan and D. Kodeboyina. Work Coordination for Grid Computing. To appear 2006. http://www.mcs.anl.gov/ gregor/papers/vonLaszewski- work-coordination.pdf
29. H.C.Lee et al.: Grid-enabled High Throughput in-silico Screening Against Influenza A Neuraminidase; NETTAB 2006, Santa Margherita di Pula, July 10-13, 2006; *submitted to a special issue of IEEE Transactions on Nanobioscience*
30. B. Lin, P. A. Dinda. VSched: Mixing Batch And Interactive Virtual Machines Using Periodic Real-time Scheduling. In *Procs. ACM/IEEE conference on Supercomputing 2005 (SC'05).* 2005
31. B. Lin, P. A. Dinda and D. Lu. User-driven Scheduling of Interactive Virtual Machines. In *Procs. of the Fifth IEEE/ACM Int. Workshop on Grid Computing (GRID'04)*, pp 380–387. 2004
32. A.Manara et al.: Integration of new communities in the Grid for mission critical applications: distributed radio-requency compatibility analysis for the ITU RRC06 conferece; *submitted to EGEE'06 Conference, 25-29 September 2006, Geneva, Switzerland*
33. J. MacLaren, R. Sakellariou, K. T. Krishnakumar, J. Garibaldi, and D. Ouelhadj. Towards Service Level Agreement Based Scheduling on the Grid. In *Procs 14th Int. Conf. on Automated Planning and Scheduling (ICAPS 04).* 2004
34. P. Mendez-Lorenzo et al.: Distributed Release Validation of the Geant4 Toolkit in the LCG/EGEE Environment; *submitted to IEEE Nuclear Science Symposium 2006*
35. I. Mirman. Going Parallel the New Way. *Desktop Computing*, 10(11), June 2006
36. J.T. Mościcki: Distributed analysis environment for HEP and interdisciplinary applications; *Nuclear Instruments and Methods in Physics Research A 502 (2003) 426-429*
37. A. Osorio, O. Traxer, S. Merran, F. Dargent, X. Ripoche, J. Ati. Real time fusion of 2D fluoroscopic and 3D segmented CT images integrated into an Augmented Reality system for percutaneous nephrolithotomies (PCNL). In *InfoRAD 2004, RSNA'04.* 2004.
38. F. Pacini and P. Kunzt. Job Description Language Attribute Specification. EGEE Tech. Rep. 555796. 2005 http://glite.web.cern.ch/glite/documentation/
39. R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Procs. of the Seventh IEEE International Symposium on High Performance Distributed Computing.* July 1998.

40. H. Rosmanith and D. Kranzlmuller. glogin - A Multifunctional, Interactive Tunnel into the Grid. In *Procs 5th IEEE/ACM Int. Workshop on Grid Computing (GRID'04)*, 2004.
41. L. G. Roberts. Beyond Moore's law: Internet growth trends. *IEEE Computer*, 33(1),pp 117-119, Jan. 2000
42. V. Servois, A. Osorio et al. A new PC based software for prostatic 3D segmentation and measurement : application to permanent prostate brachytherapy (PPB) evalution using CT and MR images fusion. In *InfoRAD RSNA 2002. 88th Annual Meeting of the Radiological Society of NorthAmerica*. Dec. 2002.
43. SGI. High-performance computers become interactive. www.sgi.com/company_info/newsroom/3rd_party/111505_starp.pdf. Jan. 2006
44. J. Shalf and E. W. Bethel Cactus and Visapult: An Ultra-High Performance Grid- Distributed Visualization Architecture Using Connectionless Protocols. *IEEE Computer Graphics and Applications*, 23(2):51–59. 2003.
45. S. Smallen, H. Casanova and F. Berman. Applying scheduling and tuning to on-line parallel tomography. In *Procs 2001 ACM/IEEE conference on Supercomputing (SC'01)*. 2001.
46. Q. Snell, M. J. Clement, D. B. Jackson and C. Gregory. The Performance Impact of Advance Reservation Meta- scheduling. *In Procs of the Workshop on Job Scheduling Strategies for Parallel Processing at IPDPS '00/JSSPP '00*, pp 137–153. Springer-Verlag. 2000.
47. A. C. Sodan, S. Doshi, L. Barsanti and Darren Taylor. Gang Scheduling and Adaptive Resource Allocation to Mitigate Advance Reservation Impact. In*Procs Sixth IEEE Int. Symp. on Cluster Computing and the Grid* , pp 649-653, IEEE Comp. Soc. Press, 2006.
48. A. Sundararaj, A. Gupta, and P. Dinda. Increasing Application Performance In Virtual Environments Through Run-time Inference and Adaptation. In *Procs of the 14th IEEE Int. Symp. on High Performance Distributed Computing (HPDC 2005)*. 2005.
49. M. Thomas, J Burruss, L Cinquini, G Fox, D. Gannon, I. Glilbert, G. von Laszewski, K. Jackson, D. Middleton, R. Moore, M. Pierce, B. Plale, A. Rajasekar, R. Regno, E. Roberts, D. Schissel, A. Seth, and W. Schroeder. Grid Portal Architectures for Scientific Applications. Journal of Physics, 16, pp 596-600. 2005.
50. V. Tola, F. Lillo, M. Gallegati and R. N. Mantegna. Cluster analysis for portfolio optimization. Preprint. 2005.
51. A. Tsaregorodtsev, V. Garonne, and I. Stokes-Rees. DIRAC: A Scalable Lightweight Architecture for High Throughput Computing. In *Procs 5th IEEE/ACM Int. Workshop on Grid Computing (GRID'04)*, 2004.