

# Performance-based Middleware Services for Grid Computing

Stephen A. Jarvis, Daniel P. Spooner, H el ene N. Lim Choi Keung,  
Justin R.D. Dyson, Lei Zhao and Graham R. Nudd

High Performance Systems Group, University of Warwick, Coventry, UK  
stephen.jarvis@warwick.ac.uk

## Abstract

*Managing resources and distributed workload across multiple administrative boundaries is a key issue in Grid computing and middleware research. Standard services that perform look-up, match-making, scheduling and staging are being developed to address this problem. As active middleware components, these services will allow users to identify and utilise appropriate resources that provide sustainable system- and user-level qualities of service.*

*This paper documents two enhanced match-making services that address the performance implications of executing a particular workload on a given set of resources. These services are based on an established performance prediction system that is employed at both the local (intra-domain) and global (multi-domain) levels to provide dynamic workload steering. These additional facilities bring about significant performance improvements, the details of which are presented with regard to the user-perceived quality of service and to the Grid resource utilisation.*

## 1. Introduction

The Grid computing paradigm [13, 18] promotes user collaboration through flexible and co-ordinated sharing of distributed resources. This includes providing access to resources that traditionally reside outside the domain of a typical user, such as a super-computing resource or specialist instrumentation. In order to support this approach, it is necessary to provide middleware services that can operate effectively within large, highly dynamic, heterogeneous multi-domain environments. Resource management in this context is compounded by the need to provide a sustainable, reliable and predictable computing service, despite variations in the available resources and user demands.

Such services must adapt to varying architectures, system characteristics and user requirements in real time, and implement *match-making* capabilities that link appropriate resources to the varying workload. While these services

may operate on system properties such as architectural, operating system and memory compatibilities, they can also be driven by soft-matching requirements such as task deadlines and user priority.

This paper documents a performance-based middleware service that provides additional match-making capabilities based on the anticipated run-time characteristics of an application. The middleware operates at two levels: at the intra-domain level where the resource management system uses predictive information to tune task execution, rearrange the task queue and maximise the utilisation of the servicing architectures; and at the multi-domain level where a peer-to-peer agent system is used to steer tasks to under-utilised resources, thereby balancing workload and improving resource usage.

A key feature of this research is that these autonomic middleware services are integrated with standard Grid middleware tools. Intra-domain resource management is implemented as an additional service that directs the Condor scheduler [19], which itself provides the deployment of the workload to the physical resources. This extra layer of management uses an iterative scheduling algorithm to select task schedules that satisfy service demands, minimise resource idle time and balance the workload over the available intra-domain resources.

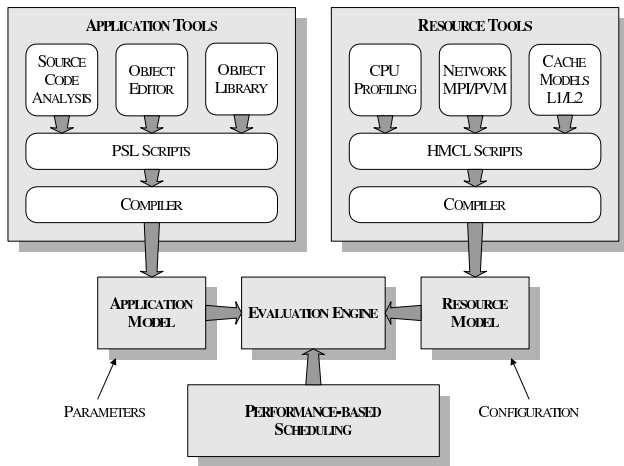
The peer-to-peer agent system is used to assist in multi-domain workload management. At this level, the middleware uses the Globus information services [12, 14] to support resource advertisement and discovery. This allows direct inter-domain task migration to occur and results in tangible performance benefits. Compelling evidence of these benefits is documented for a 256-node Grid system.

Details of the supporting performance prediction system are documented in section 2. The intra-domain resource management services are described in section 3; this is extended to multi-domain task management in section 4. Experimental results in sections 5, 6 and 7 provide evidence of substantial improvements that these additional middleware services provide in terms of quality of service and Grid resource utilisation.

## 2. The PACE Toolkit

The middleware is based on an established Performance Analysis and Characterisation Environment (PACE) [20] developed by the High Performance Systems Group at the University of Warwick. PACE provides quantitative data concerning the performance of applications running on high performance parallel and distributed computing systems. Tools are used to characterise an application using a performance modelling language that captures the key communication/computation steps. Resource capabilities are identified using hardware micro-benchmarks and the resulting hardware and application models are combined in real time to derive predictive execution data – Figure 1.

The separation of hardware and software components allows PACE to evaluate *performance scenarios*, such as the scaling effect of increasing the number of processors or the impact of modifying a processor mapping strategy. This feature is particularly useful in the context of heterogeneous environments where the same application model can be applied to numerous hardware models to obtain different run-time predictions or scaling behaviours.

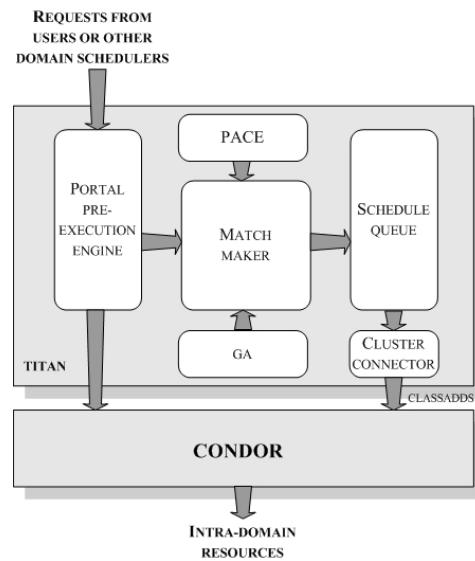


**Figure 1.** An outline of the PACE system including the application and resource modelling components and the parametric evaluation engine which combines the two.

The PACE performance evaluation and prediction capabilities have been validated using ASCI (Accelerated Strategic Computing Initiative) demonstrator applications [17]. The toolkit provides a good level of predictive accuracy and the evaluation process typically completes in seconds. PACE has been used in a number of other high-performance settings, including the performance optimisation of financial applications [21], real-time performance analysis and application steering [1] and the predictive performance and scalability modelling of the application Sweep3D [8].

## 3. Intra-domain Resource Management

The management of resources at the intra-domain level is provided by the combination of a scheduler co-ordinator Titan [23] and a standard commodity scheduler (in this case Condor [19], operated in dedicated mode rather than cycle-stealing mode). Titan makes use of the PACE predictions to manage incoming tasks and to improve resource utilisation through task packing and idle-time reduction. The goal of Titan is to reduce the run-time of applications by targeting suitable resources and scaling the application appropriately. This is not performed in isolation as the system is also sensitive to surrounding tasks and requirements. At the core of the system is a genetic algorithm (GA) which is used to balance conflicting parameters and find schedules that are best able to meet the task requirements and resource capabilities. Figure 2 provides an overview of the intra-domain level middleware components.



**Figure 2.** The key intra-domain level middleware components. Tasks that have associated performance data are processed by the Titan predictive co-scheduler. This coordinates the mapping of tasks to resources before they are finally committed to the physical hardware by Condor.

Tasks enter the system by means of a portal (currently implemented as command-line utility) from which the user specifies an application name, deadline, PACE performance model and a pre-execution script. The pre-execution script runs in a security sandbox (with minimal privileges) and allows the application to perform application initialisation, such as modifying input control files based on the processor

mapping recommended by Titan, or downloading appropriate binaries for the resource type.

On receipt of the application request, Titan combines the PACE application model with the hardware model that is appropriate for the intra-domain resources. The model is evaluated for different processor configurations to obtain a scaling graph for the application on the given resource. Comparing the application’s minimum run-time with the current queue length of the scheduler, Titan is able to predict when the application will complete and can compare this with the user-specified deadline. If the deadline of the task can be met, the task is submitted to the local domain queue for subsequent processing. If the deadline cannot be met then Titan will negotiate with other Titan co-schedulers to determine whether the task request can be satisfied by neighbouring resource domains. When it is not possible to meet the deadline, the task is run on the resource that minimises the deadline failure.

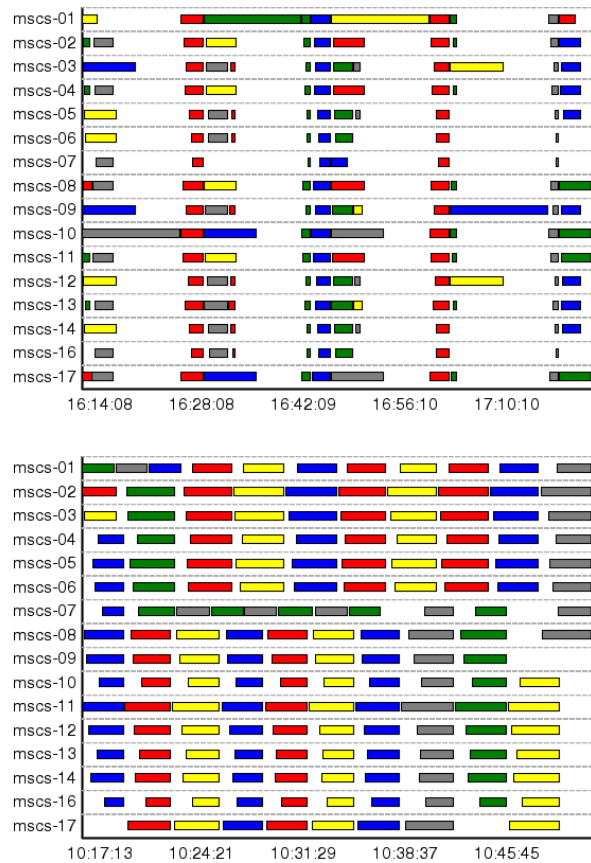
When a task is accepted for processing it is placed in Titan’s scheduling queue with the other accepted tasks. Titan’s genetic algorithm works on this queue while the jobs are waiting, exploring task mappings that reduce the makespan (end-to-end run-time), idle time (locked between processes) and average delay (the amount of time tasks complete before or after their deadline). The genetic algorithm achieves this by creating multiple scheduling solution sets, evaluating these solutions and ejecting unsuccessful schedules while maintaining the good schedules for the next generation. As better schedules are discovered, they replace the current *best schedule* and the task queue is re-ordered appropriately. This will continue up to the point at which the intra-domain resources are free to accept the tasks at the front of the scheduling queue.

As the tasks reach the front of the scheduling queue, they are removed from the queue for staging onto the receiving resources. This staging process involves tasks being presented to Condor by means of a *submit file* that details the various input, output and argument parameters required by the application. Condor allows the user to specify a requirements field that contains ClassAds [22] that are typically used to select and rank machines with specific configurations. In this work, we utilise this functionality by creating ClassAds that force Condor to co-allocate and run a task on designated machines using the name attribute. By keeping Condor’s queue empty and forcing the mapping of tasks, Titan is therefore able to direct the scheduling behaviour of the system.

Titan monitors the intra-domain resources using Condor’s status tools and can respond to resource variations such as machines going off-line or being re-introduced into the local domain. In each case, the genetic algorithm is able to respond to the changes in state and compensate accordingly [23]. This prevents the scheduling queue becoming

stuck by a task that specifies more resources than are available, and allows the system to make use of new resources as they come on-line.

When tasks successfully terminate, Titan compares the actual run-time of the task against the predicted run-time generated by PACE. This information is then fed back to the PACE evaluation engine so that future predictions of the same application on the same hardware can be improved.



**Figure 3. Top – run-time schedule using Condor, resulting in a makespan of 70.08 minutes. Bottom – run-time schedule using Condor and the predictive co-scheduler Titan, resulting in a makespan of 35.19 minutes.**

The capabilities of the predictive co-scheduler are demonstrated. This is performed by selecting 30 random tasks from a set of 5 parallel kernels and queuing the tasks onto 16 homogeneous hosts. Each of the parallel kernels has a corresponding PACE application model and the task set is chosen so that each of the tasks exhibit different parallel scaling behaviours.

The results in Figure 3 are based on run-time measurements obtained from a cluster of 16 1.4Ghz Pentium 4s with communication across a Fast Ethernet network. Using a

population size of 40, the Titan co-scheduler (running on an 800Mhz Pentium III) is capable of performing approximately 100 GA iterations per second. Each task is assigned an arbitrary deadline, all the tasks run to completion and pre-empting (the ability to multi-task or micro-schedule) is not permitted.

Table 1 shows the results of these experiments. In the first three experiments Condor is operated without the co-scheduler Titan. The tasks submitted to Condor in the first experiment are specified with an arbitrary number of hosts (from 1 to 16) for each task. This is representative of users submitting tasks without regard to the current queue or how best the tasks scale over the given resources. In many cases, larger tasks block smaller tasks (see Figure 3 – top) and this results in a large idle-time and makespan.

The second experiment illustrates a common scenario where users specify the maximum number of machines in the cluster on which to run their tasks. While in most cases this reduces the single-task execution time, the improvement over fewer hosts may in fact be marginal and blocking is still common (the run-time view is omitted for brevity).

In the third experiment the tasks are submitted with a pre-calculated number of hosts. As one would expect, this significantly reduces the make-span although it is a scheme that requires a good deal of pre-execution analysis and user cooperation.

In the final experiment, the Titan co-scheduler is used to dynamically map tasks to resources before the tasks are deployed to the physical resources by Condor (see Figure 3 – bottom). Significant improvements are made by searching for a schedule that minimises the makespan, reduces the idle time and minimises the average delay.

Experiment	Time (m)	Idle (%)
Condor		
arbitrary hosts per task	70.08	61
maximum hosts per task	69.10	28
pre-calculated hosts per task	38.05	14
Condor plus co-scheduler Titan	35.19	21

**Table 1. Experimental results using Condor and using Condor with the Titan co-scheduler.**

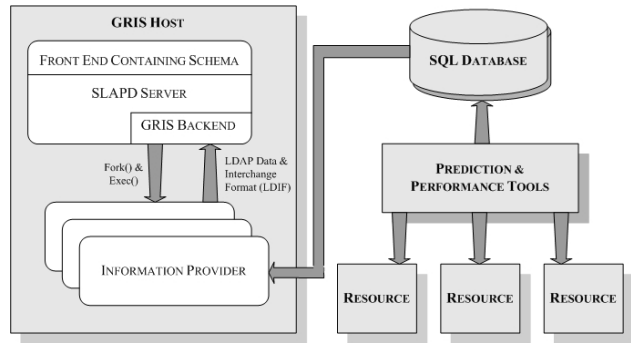
The results in Table 1 demonstrate the improvements obtained using this predictive co-scheduling technique. Over the first two experiments the Condor-Titan system effectively halves the makespan (from 70 to 35 minutes). Even when the best resource mapping and schedule is precalculated by the users (the third experiment), Condor-Titan still improves the makespan by 8%. These results are significant, but of additional interest is the ability of the predictive co-scheduling to self-manage and adapt according to additional quality of service features, see section 6.

## 4. Multi-domain Task Management

Intra-domain application performance and resource usage data is stored and made available throughout the multi-domain grid system components via a *performance information service*. The implementation of this information service is based on the Monitoring and Discovery Service (MDS) [11] from the Globus Toolkit [12]. This consists of a number of configurable information providers (Grid Resource Information Services) and configurable directory components (Grid Index Information Services).

Information about the intra-domain resources (e.g. a parallel machine or cluster of workstations) is stored at a local Grid Resource Information Services (GRIS) host. The back-end information storage is maintained by a relational database, and this information is transferred to the GRIS host through LDIF (LDAP Data Interchange Format) supported information providers – Figure 4.

Each GRIS host provides a domain-level subset of information about resources participating in the grid. Higher-level (multi-domain) access to this information is provided through the MDS Grid Index Information Services (GIIS). The advantage of this is that it provides a unified solution to the distribution of data, it is decentralised (and therefore robust) and information providers are located logically close to the entities which they describe.



**Figure 4. Implementation of a performance information service using the Globus MDS. Data is stored in a local relational database and data exchange is provided through a number of LDIF supported information providers.**

The data inside the performance information service is utilised by a network of agents which cooperate to provide multi-domain task management. The agents provide a higher-level approach to task management (than the Titan GA) that is able to deliver increased scalability and adaptability. In the current implementation, each domain is represented by a single agent and agent-level communication is used to coordinate inter-domain resource broker-

age. The network of agents is dynamic, which means that as new resources become available (or current resources go down) the middleware is able to dynamically reconfigure accordingly. The implementation of the agent system is documented in [7]; of particular interest is the ability of the agents to provide *resource advertisement* and *resource discovery* services.

$r$	$r/s$	$M$	$t(s)$	$\varepsilon(s)$	$\nu(\%)$
200	1	OFF	839	-1	24
200	1	ON	302	78	51
200	2	OFF	812	-36	25
200	2	ON	245	72	50
200	5	OFF	814	-64	24
200	5	ON	218	62	49
500	1	OFF	1784	-112	28
500	1	ON	633	78	57
500	2	OFF	1752	-205	29
500	2	ON	461	66	57
500	5	OFF	1877	-276	27
500	5	ON	305	32	68
1000	1	OFF	2752	-314	36
1000	1	ON	1160	79	61
1000	2	OFF	2606	-493	39
1000	2	ON	669	65	74
1000	5	OFF	2756	-681	36
1000	5	ON	467	-6	77

**Table 2. Experimental results:**  $r$  is the total number of requests (load);  $r/s$  is the request submission rate per second;  $M$  represents whether the predictive middleware is active;  $t$  is the makespan;  $\varepsilon$  is the average delay and  $\nu$  is the resource utilisation.

Each agent is able to query the task queue and resource utilisation in its own local information service domain. This data is made available at the multi-domain level by the agents invoking a data-pull or a data-push with other agents in the system. These two methods form the basis for service discovery: when a request enters the system the receiving agent will first evaluate whether the request can be met locally (an intra-domain query); if this is not the case then the services provided by the neighbouring resources are queried (a multi-domain query) and the request is dispatched to the agent which is able to provide the best service.

An additional feature of this system is the integration of a *performance monitor and adviser* (PMA). The PMA is capable of modelling and simulating the performance of the agent network while the system is active. Unlike facilitators or brokers in classical agent-based systems, the PMA is not

central to the rest of the agent system; it neither controls the agent hierarchy or serves as a communication centre in either the physical or symbolic sense. The PMA monitors statistical data from the agent system and simulates optimisation strategies in real time. If a better solution (to resource discovery, advertisement, agent communication, data management etc) is discovered, then it can be deployed into the live agent system (see [6] for details).

## 5. Case Study

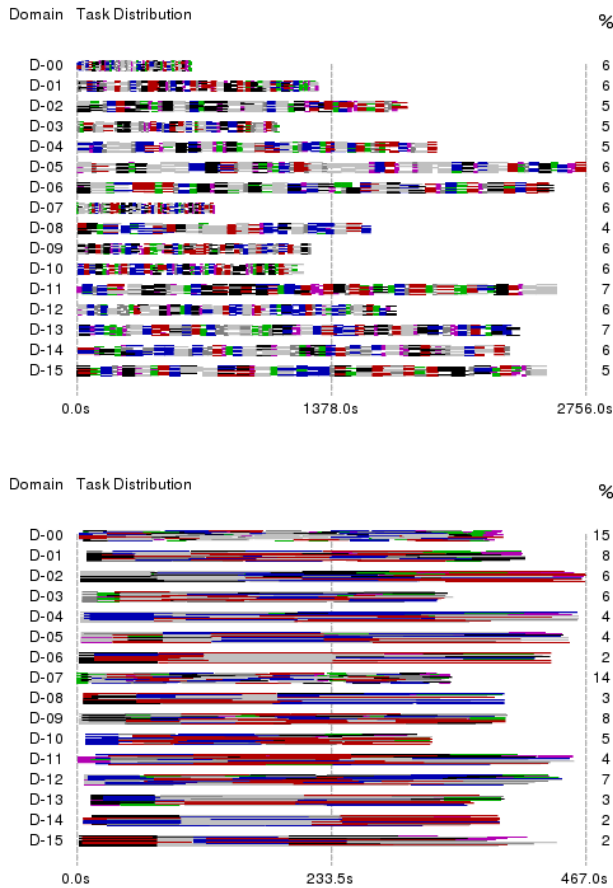
The impact of employing this middleware is simulated on a 256-node grid consisting of 16 separate resource domains. At the intra-domain level, the grid is homogeneous (containing 16 node multiprocessors or clusters of workstations); at the multi-domain level, the grid is heterogeneous (consisting of 6 different architecture types). Agents are mapped to each domain and therefore represent sub-components of the grid with varying computational capabilities.

A number of experiments are run in which 200, 500 and 1000 requests ( $r$ ) are sent to randomly selected agents at intervals of 1, 2 and 5 requests per second ( $r/s$ ); representing a broad spread of workloads and bursts of activity. During each experiment three system criteria are monitored:

- Throughput – the makespan or time to completion ( $t$ ) for all of the submitted tasks to execute – this is calculated as the maximum end time (of a task) minus the minimum start time;
- Quality of service – represented through the average delay ( $\varepsilon$ ), the amount of time tasks complete before or after their deadline. The deadlines are randomly assigned from the range of predicted values for each of the domain architectures, with suitable time allowed for staging and network latency;
- Resource utilisation – calculated as the time over which tasks are mapped to hosts. System-wide utilisation ( $\nu$ ) represents the average over each of the resource domains.

The experimental results in Table 2 represent two scenarios, when the predictive middleware ( $M$ ) is ON and when the predictive middleware is OFF. In the case when the middleware is off and the system load and submission rate are low (200 requests submitted at 1 per second), the makespan is 839 seconds. As the workload increases to 500 and 1000 tasks, so the makespan increases accordingly (to 1784 and 2752 seconds respectively). There are small variations in the makespan as the submission rate ( $r/s$ ) increases, these are not considered significant and will depend in part on the random selection of tasks for each experiment.

Of particular importance is the decrease in makespan when the predictive middleware is activated. When the system load and submission rate are low (200 requests submitted at 1 per second), the makespan is reduced by 62% (from 839 to 302 seconds). We also find that as the submission rate increases, so the improvements brought about by the middleware also increase – the makespan is reduced by 70% (for 200 tasks) with a submission rate of 2 tasks per second and by 73% with a submission rate of 5 tasks per second. This highlights the effectiveness of the multi- and intra-domain performance services. With more tasks on which to operate, each service is able to explore additional task allocation and scheduling scenarios and therefore select those mappings that bring about the best system improvements.



**Figure 5. The results of running 1000 tasks submitted at a request rate of 5 per second. Top – when the predictive middleware is inactive; bottom – when the predictive middleware is active. Note the improved makespan, 2756 seconds to 467 seconds.**

This feature is emphasised most clearly at the highest workload. When the system is subject to 1000 tasks sub-

mitted 1 per second, the middleware is able to reduce the makespan by 58% (from 2752 to 1160 seconds). As the submission rate increases, so increased improvements can be observed – a reduction in makespan of 74% at a submission rate of 2 tasks per second, and a reduction in makespan of 83% at a submission rate of 5 tasks per second. Run-time views supporting these results can be found in Figure 5. The redirection of tasks by the agent system can be seen under the % column to the right of the figure. When the middleware is off, each resource domain is assigned roughly the same percentage of tasks. When the middleware is on, tasks are directed to the resources that have spare capacity (domains D-00 and D-07, for example) and moved off domains whose resources are already over-stretched (domains D-06, D-14 and D-15, for example).

## 6. Quality of Service

There are a number of definitions of *quality of service* for distributed Grid systems consisting of non-dedicated resources. Many of these focus on the resource, for example the network (including bandwidth and latency), processor availability (including FLOPS and CPU utilisation), or additional hardware facilities such as disks etc [15]. Quality of service in this research symbolises a *user-side* service that is based on the deadline assigned to each task. This has different characteristics from *advance* or *right-now* reservation where the emphasis is on the user choosing a set of resources that have been pre-selected as being able to meet their needs. The approach used here is also compatible with the new GRAM-2 [10] terminology; it deals with task SLAs rather than resource SLAs or reservation, and the mapping of tasks to resources – the binding SLA – is performed through the Titan genetic algorithm.

In this work we draw on web services research where users, or groups of users, are assigned service classes under which contractual service agreements are guaranteed for an associated cost [2]. The QoS emphasis here is on the service, rather than the tools or resources needed to deliver that service; the delivery itself is transparent and is handled by the supporting active middleware. In the remainder of this section, quality of service is analysed on a per-task and then system basis. It is equally feasible to group tasks to classes or priorities of user, the net effect would be the same.

The mapping of tasks to resources using the Titan predictive co-scheduler is also different from previous work on online-mode mapping, where tasks are assigned to resources as soon as they are received, and batch-mode mapping, where sets of tasks are collected and then manipulated before being dispatched to the underlying resources. Titan is configured with a small amount of staging time (this can be seen in the bottom run-time view of Figure 5) so that 100 iterations of the genetic algorithm are run before the tasks

begin being passed to Condor. This provides an interaction between the on-line task submission (which is driven by the availability of the resources) and the batch processing by the genetic algorithm (that operates for as long as is possible until the task at the front of the queue is dispatched).

The Titan task mapping process also differs from other QoS-based scheduling strategies. In [16] the QoS-guided MinMin mapping process first assigns tasks with high QoS requirements to resources before dealing with those task requests that have a lower priority. While this might seem like a good strategy, it is possible to use lower priority jobs to pack the tasks (see Figure 3), using spare resources for low priority tasks as soon as the tasks and resources become available. By dealing with the lower priority tasks as soon as possible, we find that they are less likely to impact on the higher priority tasks later in the schedule.

Workload	$r$	$r/s$	$M$	$\mathcal{D}$ (%)
Low	200	1	OFF	57
	200	1	ON	89
<i>improvement</i>				32
Medium	500	2	OFF	27
	500	2	ON	83
<i>improvement</i>				56
High	1000	5	OFF	9
	1000	5	ON	50
<i>improvement</i>				41

**Table 3. Percentage of tasks meeting their deadlines under low, medium and high workloads. The results represent when the middleware off and on; the results also show the percentage improvement made by activating the middleware.**

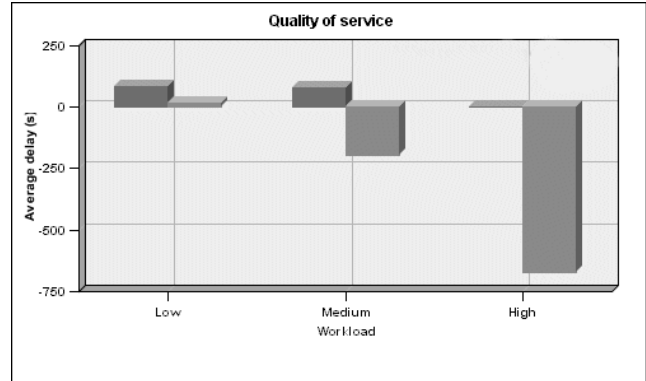
The case study in section 5 provides two different aspects of deadline-based quality of service: these are the time in which tasks complete before or after their deadline, the average delay  $\epsilon$ , and the number of tasks that complete before their deadline, termed  $\mathcal{D}$  and measured as a percentage of the total.

It can be seen in Table 3 that as the workload on the system increases, so the percentage of tasks that meet their deadline ( $\mathcal{D}$ ) decreases. The selection of deadlines in this case study is deliberately tight so that under a low workload (200 requests submitted 1 per second) 57% of the tasks complete before their assigned deadline. This decreases to 27% of tasks under a medium workload (500 requests submitted 2 per second), and to only 9% under a high workload (1000 requests submitted 5 per second).

The middleware improves  $\mathcal{D}$  by between 32 and 56%, ensuring that 89% of the tasks meet their deadlines when the workload is low. This figure drops as the workload in-

creases, to 83% at a medium workload and to 50% at a high workload; this decrease is not unexpected.

$\mathcal{D}$  provides detail on the percentage of tasks which meet their deadline, but it does not give any indication as to the degree by which this takes place. We therefore use the average delay ( $\epsilon$ ) as an additional measure of quality of service, and in so doing gain a greater insight in to the extra schedule capacity which the activation of the predictive middleware services can provide.



**Figure 6. The average delay under varying system loads. The bar to the right represents the delay when the middleware is off; the bar to the left represents the delay when the middleware is on.**

In the case when the middleware is off and the system load and submission rate are low, the average delay  $\epsilon$  is -1 second, see Table 2. As the submission rate increases, so  $\epsilon$  increases to -36 seconds at a submission rate of 2 requests per second and to -64 seconds at 5 requests per second; this trend is also demonstrated at the higher workloads. Figure 6 shows the combined average delay for the low, medium and high workloads. The bar to the right represents the average delay when the middleware is inactive.

Activating the predictive middleware has a positive effect on  $\epsilon$ ; when 200 requests are sent at 1 per second,  $\epsilon$  is 78 seconds, indicating spare schedule capacity. When the workload and submission rate are high (1000 requests at 5 per second) the impact is marked; rather than running 11 minutes over schedule (-681 seconds), the prediction-based middleware is able to reduce this to -6 seconds. These results can be seen in Figure 6, where the bar to the left represents the average delay when the middleware is active.

It can be seen that without the additional active middleware services, the quality of the system rapidly deteriorates, both from the point of view of the number of tasks that meet their deadlines and also the extra capacity which is available. With the middleware enabled, the user-side quality of service is maintained up to the point at which the system becomes fully loaded.

## 7. Resource Usage

The high-level task migration provided by the agent system delivers basic load balancing across the resources in the underlying grid. This redirection of tasks is different from that provided by GrADS [4], which allows the migration of running tasks in response to system load in order to improve performance and prevent system degradation. In the Titan system, once tasks have been staged, they run to completion. This method essentially moves all the decision making forward (to pre-staging) and as a result has negligible run-time overheads as no additional checkpointing is needed; all run-time reporting at the co-scheduler level is done using Condor’s status tools and as such no additional functionality is therefore required<sup>1</sup>. Although this system shares many of the fundamental properties of the NetSolve Environment [3, 9], its resource management also differs in a number of ways; in particular, PACE is non-intrusive and the predictive data which it supplies does not require any link between a client library and the application itself.

Figure 7 shows the resource utilisation ( $\nu$ ) for the system under low, medium and high workloads. As in Figure 6, the bar to the left represents the case when the middleware is active and the bar to the right represents the case when the middleware is inactive. As can be expected, the resource utilisation increases as the system load increases (both when the middleware is active and inactive). What is significant is how the middleware improves the utilisation, by 28% for the low workload, 31% for the medium workload and 40% for the high workload. Also of note is the fact that these improvements increase as the workload increases.

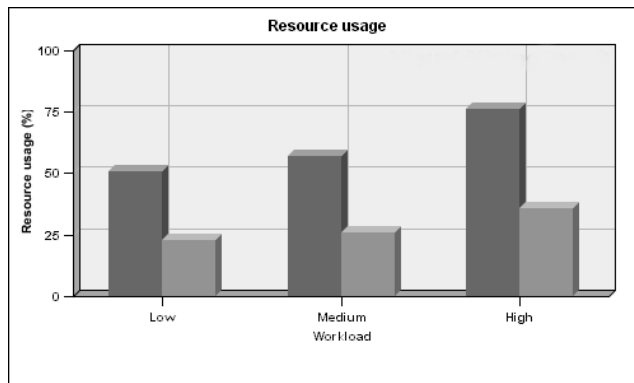
If we analyse the resource usage at the intra-domain level, we find that the difference in resource utilisation (between the middleware being active and inactive) is larger in the domains with the highest computational capabilities (for example a 40% difference for D-00 and D-07, and a 4% difference for domains D-06 and D-15). These results are caused by the predictive middleware being able to identify and make use of the larger proportion of idle time on the higher capacity resources. This trend is found to be uniform across the different workloads.

Of additional interest is the relationship between the quality of service measures and resource utilisation. The results in Table 3 and Figures 6 and 7 allow the point at which the middleware begins to fail the majority of users to be identified. This can be seen in Figure 6 when  $\varepsilon$  switches from being positive to negative, and in Table 3 when  $\mathcal{D}$  drops to 50%. This system state corresponds to the case when the resource utilisation measures 75%.

We are able to choose some combination of these met-

<sup>1</sup>This may not suit the management of very long running tasks, but it does provide a good cost-benefit for the type of scientific applications that are run in this case study.

rics and some suitable thresholds under which to determine when the system has reached *user-side* capacity. This represents the point at which we can predict when a task will not meet its deadline, even before the task has been deployed or any additional service discovery has been instantiated. Similarly, we can predict what computing reserves are available and know at which point new facilities will need to be interconnected to meet the increasing demands on the Grid. These provisioning and capacity planning features may also link with other autonomic services (e.g. [5]) which are able to determine *time of use* meters to reflect variations in price-time processing, and are the subject of future work.



**Figure 7. The resource usage under varying system loads. The bar to the right represents the resource usage when the middleware is off; the bar to the left represents the resource usage when the middleware is on.**

## 8. Conclusions

Autonomic middleware services are set to play an increasingly important role in the management of resources and distributed workloads in emerging wide-area, heterogeneous distributed computing environments. This paper documents two supporting performance middleware services for these architectures, which are based on existing Condor- and Globus-enabled Grid infrastructures.

This paper details a local (intra-domain) level predictive co-scheduler, that uses performance prediction data generated by the PACE toolkit to support intra-domain task management. This service is extended to the global (multi-domain) level through an information service based on the Globus MDS. The global middleware uses a peer-to-peer agent system and high-level workload steering strategies to balance system load and improve system-wide resource utilisation.

The improvements brought about by these performance-

based middleware services are significant. At the intra-domain level, predictive co-scheduling can halve the makespan of a set of typical scientific tasks running on an Condor cluster. At the multi-domain level, the combination of predictive co-scheduling and agent-based task migration improves the makespan by up to 83% in a heavily loaded 256-node Grid. These improvements also manifest themselves in the recorded resource usage and also the associated user-side quality of service.

A multi-tiered approach to autonomic performance-service provision is likely to prove successful, where improvements brought about at a local level can be exploited by cooperative wide-area management tools.

## Acknowledgements

This work is sponsored in part by grants from the NASA AMES Research Center (administrated by USARDSG, contract no. N68171-01-C-9012), the EPSRC (contract no. GR/R47424/01) and the EPSRC e-Science Core Programme (contract no. GR/S03058/01).

## References

- [1] A. Alkindi, D. Kerbyson, and G. Nudd. Optimisation of application execution on dynamic systems. *Future Generation Computer Systems*, 17(8):941–949, 2001.
- [2] J. Aman, C. Eilert, D. Emmes, P. Yacom, and D. Dillenger. Adaptive algorithms for managing a distributed data processing workload. *IBM Systems Journal*, 36(2):242–283, 1997.
- [3] D. Arnold, and J. Dongarra. The NetSolve Environment: Processing Towards the Seamless Grid. *29th International Conference on Parallel Processing (ICPP-2000)*, Toronto Canada, August 21–24, 2000.
- [4] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, and R. Wolski. The GrADS Project: Software Support for High-Level Grid Application Development. *International Journal of High Performance Computing Applications*, 15(4):327–344, 2001.
- [5] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic models for resource management and scheduling in Grid computing. *Concurrency and Computation: Practice and Experience*, 14:1507–1542, 2002.
- [6] J. Cao, S. Jarvis, S. Saini, D. Kerbyson, and G. Nudd. ARMS: an agent-based resource management system for grid computing. *Scientific Programming, Special Issue on Grid Computing*, 10(2):135–148, 2002.
- [7] J. Cao, D. Kerbyson, and G. Nudd. High performance service discovery in large-scale multi-agent and mobile-agent systems. *International Journal of Software Engineering and Knowledge Engineering, Special Issue on Multi-Agent Systems and Mobile Agents*, 11(5):621–641, 2001.
- [8] J. Cao, D. Kerbyson, E. Papaefstathiou, and G. Nudd. Performance modelling of parallel and distributed computing using PACE. *Proceedings of 19th IEEE International Performance, Computing and Communication Conference (IPCCC'00)*, pp. 485–492, 2000.
- [9] H. Casanova, and J. Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. *International Journal of Supercomputer Applications and High Performance Computing*, 11(3):212–223, 1997.
- [10] K. Czajkowski. GT Resource Management and Scheduling: The Globus Perspective *GlobusWorld 2003*, January 13–17 2003, San Diego, CA, USA.
- [11] S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. *10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10 '01)*, 2001.
- [12] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.
- [13] I. Foster and C. Kesselman. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1998.
- [14] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. Grid services for distributed system integration. *IEEE Computer*, 35(6):37–46, 2002.
- [15] I. Foster, A. Roy, and V. Sander. A Quality of Service Architecture that Combines Resource Reservation and Application Adaption. *Proceedings of 8th International Workshop on Quality of Service*, pp. 181–188, June 5–7 2000, Pittsburgh, PA, USA.
- [16] X. He, X. Sun, and G. Laszewski. A QoS Guided Scheduling Algorithm for Grid Computing. *International Workshop on Grid and Cooperative Computing (GCC02)*, Hainan, China, 2002.
- [17] D. Kerbyson, H. Alme, A. Hoisie, F. Petrini, H. Wasserman, and M. Gittings. Predictive performance and scalability modelling of a large-scale application. *Proceedings of Supercomputing '01*, 2001.
- [18] W. Leinberger and V. Kumar. Information power grid : The new frontier in parallel computing? *IEEE Concurrency*, 7(4), 1999.
- [19] M. Litzkow, M. Livny, and M. Mutka. Condor – a hunter of idle workstations. *Proceedings of 8th International Conference on Distributed Computing Systems (ICDCS'88)*, pp. 104–111, 1988.
- [20] G. Nudd, D. Kerbyson, E. Papaefstathiou, S. Perry, J. Harper, and D. Wilcox. PACE : A toolset for the performance prediction of parallel and distributed systems. *International Journal of High Performance Computing Applications*, 14(3):228–251, 2000.
- [21] S. Perry, R. Grimwood, D. Kerbyson, E. Papaefstathiou, and G. Nudd. Performance optimisation of financial option calculations. *Parallel Computing*, 26(5):623–639, 2000.
- [22] R. Raman, M. Livny, and M. Solomon. Matchmaking: Distributed Resource Management for High Throughput Computing. *7th International Symposium on High Performance Distributed Computing (HPDC-7 '98)*, 1998.
- [23] D. Spooner, S. Jarvis, J. Cao, S. Saini, and G. Nudd. Local grid scheduling techniques using performance prediction. *IEE Proceedings: Computers and Digital Techniques*, 2003.