Available online at www.sciencedirect.com

SCIENCE DIRECT®

Future Generation Computer Systems 22 (2006) 745–754

FGCS
FUTURE GENERATION COMPUTER SYSTEMS

www.elsevier.com/locate/fgcs

Performance prediction and its use in parallel and distributed computing systems☆

Stephen A. Jarvis[a,*], Daniel P. Spooner[a], Helene N. Lim Choi Keung[a], Junwei Cao[b], Subhash Saini[c], Graham R. Nudd[a]

[a] *Department of Computer Science, University of Warwick, Warwick CV4 7AL, UK*
[b] *Center for Space Research, Massachusetts Institute of Technology, Cambridge, MA, USA*
[c] *NASA Ames Research Center, Moffett Field, CA, USA*

Available online 2 May 2006

Abstract

Performance prediction is set to play a significant role in supportive middleware that is designed to manage workload on parallel and distributed computing systems. This middleware underpins the discovery of available resources, the identification of a task's requirements and the match-making, scheduling and staging that follow.

This paper documents two prediction-based middleware services that address the 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 system- and user-level qualities of service. The middleware has been designed for the management of resources and distributed workload across multiple administrative boundaries, a requirement that is of central importance to grid computing.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Performance prediction; Resource management; Grid computing

1. Introduction

The computing architectural landscape is changing. High-end resources that were once large, multi-processor supercomputing systems are being increasingly replaced by heterogeneous commodity PCs and complex powerful servers. These new architectural solutions, including the Internet computing model [28] and the grid computing [18,25] paradigm, aim to create integrated computational and collaborative environments that provide technology and infrastructure support for the efficient use of remote computing facilities. The adoption of such architectures rests on the outcome of a number of important research areas; one of these – *performance* – is fundamental, as the uptake of these approaches relies on their ability to provide a steady and reliable source of capacity and capability computing power.

While the study of performance in relation to computer hardware and software has been a topic of much scrutiny for a number of years, it is likely that this research area will change to reflect the emergence of geographically dispersed networks of computing resources such as grids. There will be an increased need for high performance resource allocation services [3] and an additional requirement for increased system adaptability in order to respond to the variations in user demands and resource availability. Performance engineering in this context raises a number of important questions, the answers to which will impact on the utilisation and effectiveness of related performance services:

What does the performance data describe? The response of a system (or user) to the performance data will depend on the nature of the data. This might include timing data for the run-time of a particular application (on a given resource) or data relating to the monitoring of various network

☆ Sponsored in part by grants from the NASA AMES Research Center (USARDSG N68171-01-C-9012), the EPSRC (GR/R47424/01) and the EPSRC e-Science Core Programme (GR/S03058/01).

* Corresponding author. Tel.: +44 2476 524258; fax: +44 2476 573024.
E-mail address: stephen.jarvis@warwick.ac.uk (S.A. Jarvis).

0167-739X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2006.02.008

and computational resources, for example the communication latencies provided by network monitors such as NWS [35].

How is this performance data obtained? Gathering performance data can be achieved by a number of methods. Monitoring services provide records (libraries) of dynamic information such as resource usage or characteristics of application execution. This data can be used as a benchmark for anticipating the future performance behaviour of an application, a technique that can be used to extrapolate a wide range of predictive results [15]. Alternatively it is possible to extract data from an application through the evaluation of analytical models. While these have the advantage of deriving a priori performance data – the application need not be run before performance data can be collected [1,29] – they are offset by the complexity of model generation.

How is this data classified? Monitored data is often fixed-scenario – based on a particular run on a particular machine – in contrast, analytical approaches can produce parametric models which allow the investigation of performance scenarios through extrapolation. Data will also relate to different levels of abstraction in the system; this may include different application software components [34] or different machine instruction benchmarks [23] for example.

How can this data be used? Once the data has been obtained it needs to be published. This can be achieved through *information services* that ensure the shared performance data remains current amongst the distributed nodes in the system. The delivery of the data via information services allows it to be used for resource scheduling [7,33], batch queueing [27], resource discovery [10,11] or resource brokerage [8,21]. The data might also be used to manage workload from the point of view of service contracts, deadlines, or other user and system defined QoS (Quality of Service) constraints.

What will acting on this data achieve? The provision of performance information can have a number of benefits: distributed performance services [19] can be built that allow middleware to steer tasks to suitable architectures; the QoS demands of users can be serviced in resource efficient ways; the architecture can be configured so that the best use is made of its resources; the capabilities of the architecture can be extended, and configurations for providing application steering can be implemented.

This paper addresses these issues in the context of an application performance prediction environment. The *Performance Analysis and Characterisation Environment* (PACE) [29] developed by the High Performance Systems Group at the University of Warwick is a performance prediction system that provides quantitative data concerning the performance of (typically scientific) applications running on high performance parallel and distributed computing systems. The system works by characterising the application and the underlying hardware on which the application is to be run, and combining the resulting models to derive predictive execution data. PACE provides the capability for the rapid calculation of performance estimates without sacrificing performance accuracy. PACE also offers a mechanism for evaluating performance scenarios – for example the scaling effect of increasing the number of processors – and the impact of modifying the mapping strategies (of process to processor) and underlying computational algorithms [12].

The real-time capabilities and parametric prediction functions (see Section 2) allow PACE to be used for the provision of dynamic performance information services. These in turn can be used to aid the scheduling of tasks over clusters of homogeneous resources (see Section 3), and provide a basis for the higher-level management of grid system resources (see Section 4). Results (in Section 5) show that employing prediction techniques at these two system levels provides an efficient framework for the management and distribution of multiple tasks in a wide-area, heterogeneous distributed computing environment.

2. The PACE toolkit

Details of the PACE toolkit can be seen in Fig. 1. An important feature of the design is that the application and resource modelling are separated and there are independent tools for each.

The *Application Tools* provide a means of capturing the performance aspects of an application and its parallelisation strategy. Static source code analysis forms the basis of this process, drawing on the control flow of the application, the frequency at which operations are performed, and the communication structure. The resulting performance specification language (PSL) scripts can be compiled to an application model. Although a large part of this process is automated, users can modify the performance scripts to account for data-dependent parameters and also utilise previously generated scripts stored in an object library.

The capabilities of the available computing resources are modelled by the *Resource Tools*. These tools use a hardware modelling and configuration language (HMCL) to define the performance of the underlying hardware. The resource tools contain a number of benchmarking programs that allow the performance of the CPU, network and memory components of a variety of hardware platforms to be measured. The HMCL scripts provide a resource model for each hardware component in the system, since these models are (currently) static, once a model has been created for a particular hardware, it can be archived and reused.

Once the application and hardware models have been generated, they can be evaluated using the PACE *Evaluation Engine*. PACE allows: time predictions (for different systems, mapping strategies and algorithms) to be evaluated; the scalability of the application and resources to be explored; system resource usage to be predicted (network usage, computation, idle time etc.), and predictive traces to be generated through the use of standard visualisation tools.

The PACE performance evaluation and prediction capabilities have been validated using ASCI (Accelerated Strategic Computing Initiative) high performance demonstrator applications [12,24]. The toolkit provides a good level of predictive accuracy (an approximate 5% average error) and the evaluation process typically completes in a matter of seconds. PACE

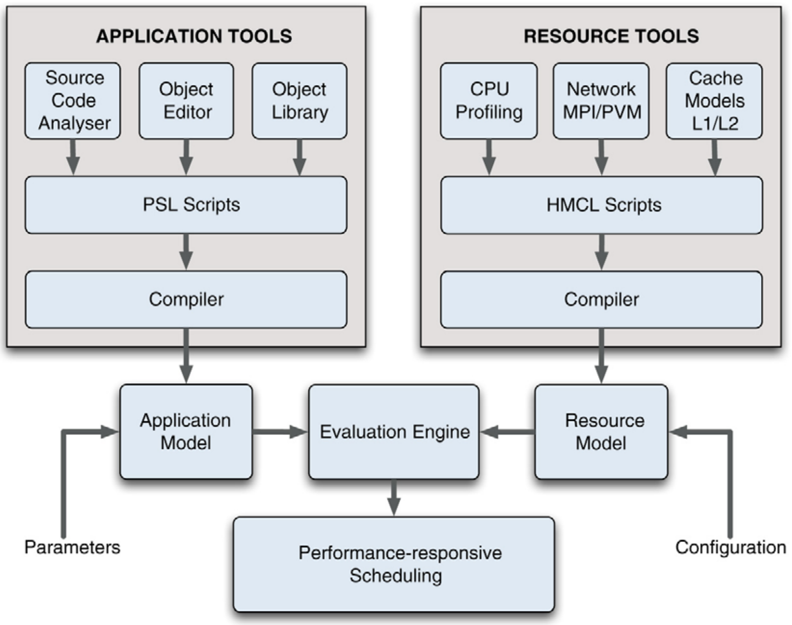

Fig. 1. An outline of the PACE system including the application and resource modelling components and the parametric evaluation engine which combines the two.

has been used in a number of other high-performance settings; these include the performance optimisation of financial applications [30], real-time performance analysis and application steering [2] and the predictive performance and scalability modelling of the application Sweep3D [12].

This work is different from our previous research in that the prediction data (from PACE) is integrated in and applied to a dynamic workload steering environment. To enable such an application, new techniques have been devised that allow PACE performance data to be generated, published and queried in real time. This provides the basis for the services documented in Sections 3 and 4.

This paper also documents two levels of performance management (intra-domain and multi-domain) based on performance data. These services are supported by abstract and domain views of prediction data provided through an integration of PACE with the Monitoring and Discovery Service (MDS) [16] from the Globus Toolkit [17].

Finally this paper demonstrates a new mapping between PACE prediction data and high-level system metrics, including average system delay, idle time and resource utilisation.

3. Intra-domain management

The management of resources at the intra-domain level is provided by the combination of a co-scheduler Titan [32] and a standard commodity scheduler (in this case Condor [27], operated in dedicated mode rather than cycle-stealing mode). Titan employs the PACE predictions to manage the incoming tasks and improve resource utilisation by coupling application performance data with a genetic algorithm (GA). The objective of the GA is to minimize the run-time of applications, reduce the resource idle time and maintain the service contracts (deadline) of each task. This is achieved by targeting suitable resources and scaling the applications using the evaluated performance models.

Titan uses a *cluster-connector* that instructs the underlying cluster management software to execute tasks in a particular order with predetermined resource mappings. This approach allows the predictive information obtained from PACE to drive the task execution provided by established workload management systems. In the case of Condor, this is achieved by generating specific resource advertisements (ClassAds [31]) that instruct Condor to run a particular task on a designated resource (or set of resources) and generating a custom submit file that details the various input, output and argument parameters as required. Execution commands are issued to the cluster manager *just-in-time*, ensuring that the task queue is not reordered by Condor before the tasks are deployed to the underlying resources. Fig. 2 provides an overview of the intra-domain level middleware components.

An advantage of the co-scheduler is that when predictive data is not available, the task can simply pass to Condor directly. The architecture does not therefore dictate a sea change in the choice of platform that is required in order to benefit from these techniques.

The cluster connector monitors the 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. This prevents the schedule queue becoming blocked by a task that specifies more resources than are available and allows the system to utilise new resources as they come on-line.

Tasks enter the system by means of a portal from which the user specifies an application name, deadline and PACE performance model. A pre-execution script can also be specified, which allows application-specific initialisation such as modifying input control files based on the processor mapping

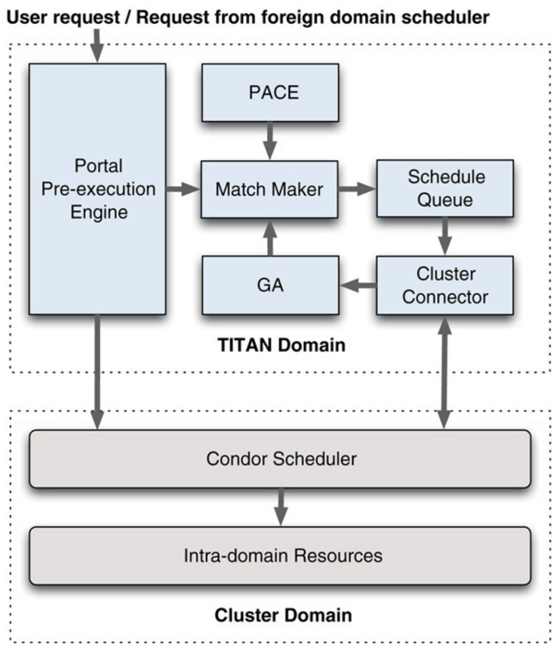

Fig. 2. Intra-domain level middleware components. Tasks that have associated performance data are processed by the Titan co-scheduler. This maps the tasks to the resources before they are finally committed to the physical hardware by Condor.

recommended by Titan, or down-loading appropriate binaries for the resource type.

Titan combines the PACE application model with the hardware model for a particular resource. The combined model is evaluated for different processor configurations to obtain a scaling graph for the application on the given resource. By comparing the application's minimum run-time with the run-time of the existing queued tasks, 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 queue for processing. If the deadline cannot be met then Titan will negotiate with other co-schedulers to determine whether the task request can be satisfied by neighbouring resource domains (see Section 4). If it is not possible to meet the deadline, the task is submitted to 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. The 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 GA creates multiple scheduling solutions, evaluates these and then rejects unsuccessful schedules while maintaining the good schedules for the next generation. As better schedules are discovered, they replace the current *best schedule* and the queue is reordered appropriately. When resources are free to accept the tasks at the front of the queue, the tasks are despatched by the cluster connector.

On task completion, Titan compares the actual run-time of the task against the predicted run-time generated by PACE, feeding back refinements where possible.

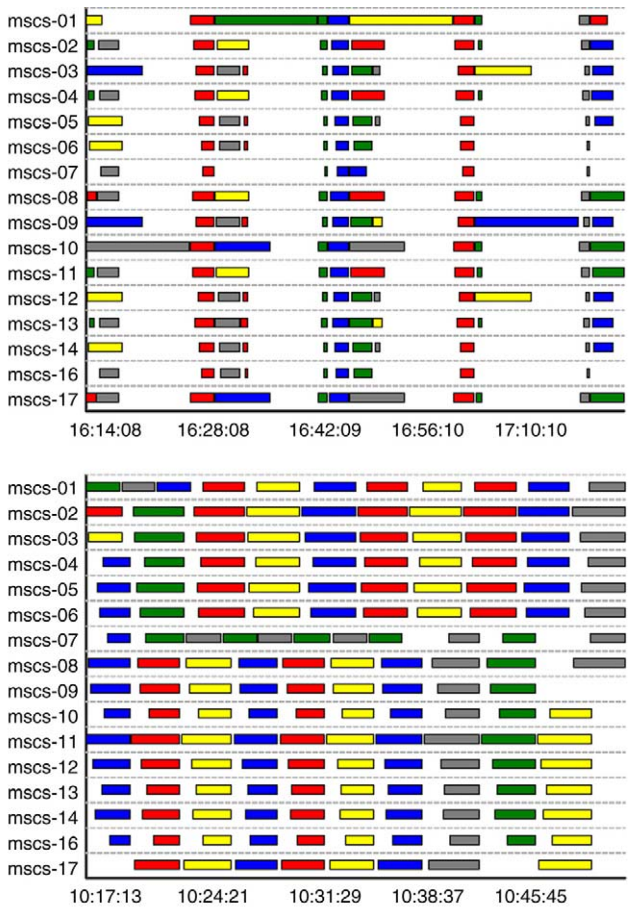

Fig. 3. Top — run-time schedule using just Condor (70.08 min); Bottom — run-time schedule using Condor and the predictive co-scheduler Titan (35.19 min).

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 Fig. 3 are based on run-time measurements obtained from a cluster of 16 1.4 GHz Pentium 4s with communication across a Fast Ethernet network. Using a population size of 40, the Titan co-scheduler (running on an 800 MHz 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 Fig. 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

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

Experiment	Time (min)	Idle (%)
Condor		
Arbitrary hosts per task	70.08	61
Maximum hosts per task	69.10	28
Calculated hosts per task	38.05	14
Condor & co-sched. Titan	35.19	21

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 Fig. 3 — bottom). Significant improvements are made by searching for a schedule that minimises the makespan, reduces the idle time and minimises the average delay, in the context of the other tasks that are currently executing on the cluster.

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 min). Even when the best resource mapping and schedule is pre-calculated 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 5).

4. Multi-domain management

To schedule across multiple grid resources with an agreed quality of service, the Titan architecture employs agent brokers that store and disseminate resource and application data. Where the resources reside outside the administrative domain, the agents communicate through existing grid information and task management services.

Each Titan scheduler is represented by an agent that promotes the capabilities of the available resource. The agent receives additional service information from other local agents that is then organised in Agent Capability Tables (ACTs). The ACTs form the basis of a *performance information service*, which is implemented as a series of Titan-specific information providers to the Monitoring and Discovery Service (MDS) [16] from the Globus Toolkit [17].

The MDS consists of Grid Resource Information Services (GRIS) and Grid Index Information Services (GIIS) that can be configured to propagate service information across Grid domains [26]. The GRIS uses an OpenLDAP server back-end

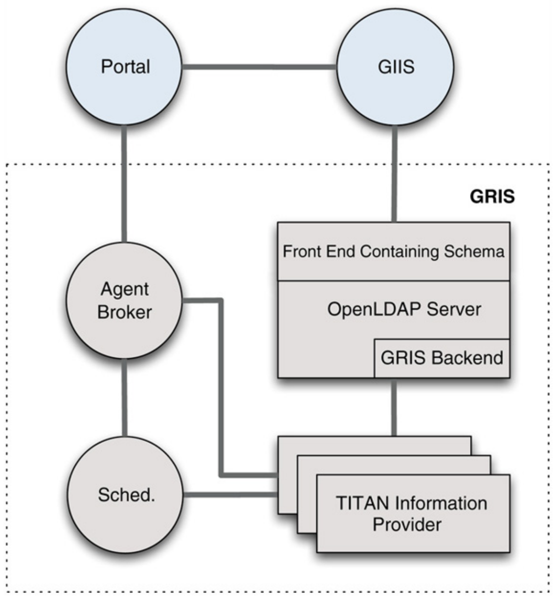

Fig. 4. Interconnect of Titan and the MDS-based performance information service.

which is customisable using extensible modules (information providers) as shown in Fig. 4. Data that is exposed to these servers is subsequently cached and propagated to parent GIIS systems using predetermined configuration rules.

The resource schedulers and agents each bind to a GRIS. This allows the inspection of current resource capabilities by the local scheduler and also by other local agents. Higher-level (multi-domain) access to this information is provided through the 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.

Agents use the information from neighbouring agents (through advertisement) or from the information service (through discovery) to deliver improved scalability and adaptability. Each domain in the current implementation is represented by a single agent and agent-level communication is used to coordinate inter-domain resource sharing. 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. The network of agents is dynamic, so as new resources become available (or current resources go down) the middleware is able to reconfigure accordingly. The implementation of the agent system is documented in [11].

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

Table 2
Experimental results: r is the 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; ε is the average delay and ν is the resource utilisation

r	r/s	M	t (s)	ε (s)	ν (%)
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

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 [10] for details).

5. Case study

The impact of employing this middleware is tested on a 256-node experimental grid. The grid consists of 16 different resource domains, each of which contains different processing capabilities. At the intra-domain level, the grid is homogeneous (each domain contains either a single 16-node multiprocessor or a commodity cluster of workstations); at the multi-domain level, the grid is heterogeneous (the entire system consists of 6 different architecture types). Agents are mapped to each domain and therefore represent sub-components of the grid with varying computational capabilities.

This experimental environment is easily reconfigured to represent a multi-cluster (or a grid of multi-clusters) or a grid/multi-cluster containing clusters of heterogeneous resources. The current configuration represents one typical scientific computational environment, governed as much by the resources (and resource domains) available to us as by any particular flavour of computational architecture.

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 (ε), 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 (ν) 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 s. As the workload increases to 500 and 1000 tasks, so the makespan increases accordingly (to 1784 and 2752 s 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 s). 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-prediction 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.

This feature is emphasised most clearly at the highest workload. When the system is subject to 1000 tasks submitted 1 per second, the middleware is able to reduce the makespan by 58% (from 2752 to 1160 s). As the submission rate increases, so greater 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 Fig. 5. The redirection of tasks by the agent system can be seen under the Task % 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 A1 and A8, for example) and moved off domains whose resources are already over-stretched (domains A6, A7, A12 and A16, for example).

5.1. 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

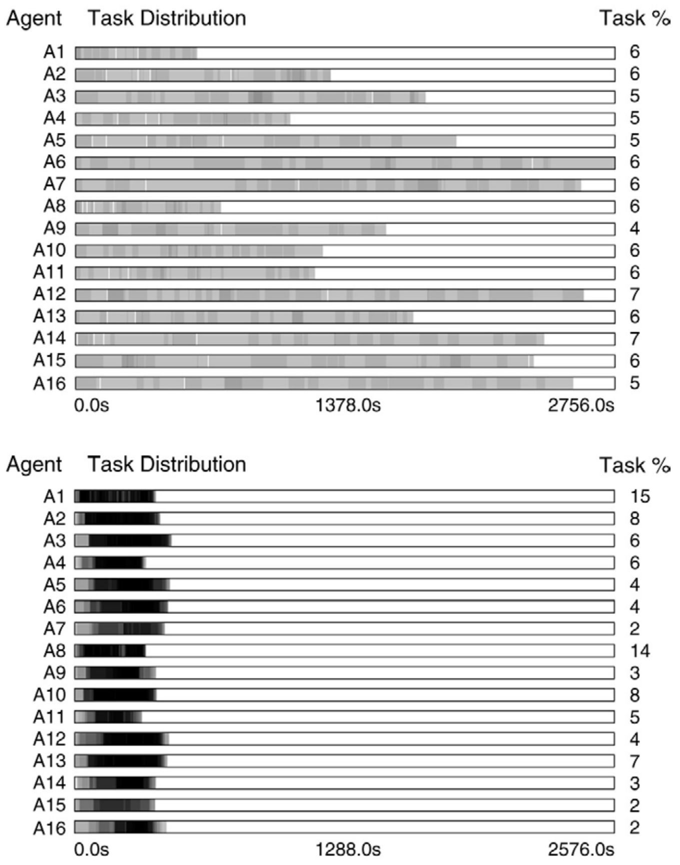

Fig. 5. The results of running 1000 tasks submitted at a request rate of 5 per second. Top — when the predictive middleware is inactive; bottom — when the predictive middleware is active. Note the improved makespan, 2756 s to 467 s. Darker shading indicates greater utilisation.

(including FLOPS and CPU utilisation), or additional hardware facilities such as disks etc. [20]. 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 [14] 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 service research where users, or groups of users, are assigned service classes under which contractual service agreements are guaranteed for an associated cost [4]. 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 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 despatched 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 Fig. 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).

Table 3
Percentage of tasks meeting their deadlines under low, medium and high workloads

Workload	r	r/s	M	δ (%)
Low	200	1	OFF	57
	200	1	ON	89
			improvement	32
Medium	500	2	OFF	27
	500	2	ON	83
			improvement	56
High	1000	5	OFF	9
	1000	5	ON	50
			improvement	41

The results represent when the middleware is off and on; the results also show the percentage improvement made by activating the middleware.

The Titan task mapping process also differs from other QoS-based scheduling strategies. In [22] 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 Fig. 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.

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 ε, and the number of tasks that complete before their deadline, termed δ 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 (δ) 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 δ 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 increases, to 83% at a medium workload and to 50% at a high workload; this decrease is not unexpected.

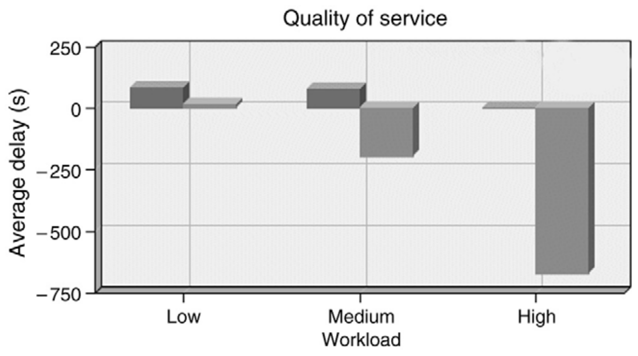

Fig. 6. The average delay under varying system loads. The bar to the right represents the delay when the middleware is off; the bar to the left represents the delay when the middleware is on.

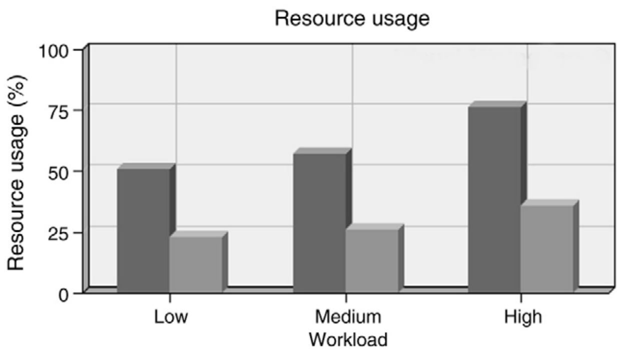

Fig. 7. The resource usage under varying system loads. The bar to the right represents the resource usage when the middleware is off; the bar to the left represents the resource usage when the middleware is on.

δ provides details 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 (ε) as an additional measure of quality of service, and in so doing gain a greater insight into the extra schedule capacity which the activation of the predictive middleware services can provide.

In the case when the middleware is off and the system load and submission rate are low, the average delay ε is -1 s, see Table 2. As the submission rate increases, so ε increases to -36 s at a submission rate of 2 requests per second and to -64 s at 5 requests per second; this trend is also demonstrated at the higher workloads. Fig. 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 ε; when 200 requests are sent at 1 per second, ε is 78 s, 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 min over schedule (-681 s), the prediction-based middleware is able to reduce this to -6 s. These results can be seen in Fig. 6, where the bar to the left represents the average delay when the middleware is active.

It can be seen that without the additional performance-based 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.

5.2. 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 [6], 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.[1] Although this system shares many of the fundamental properties of the NetSolve Environment [5,13], 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.

Fig. 7 shows the resource utilisation (ν) for the system under low, medium and high workloads. As in Fig. 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 the 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 A1 and A8, and a 4% difference for domains A6 and A16). 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 Figs. 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 Fig. 6 when ε switches from being positive to negative, and in Table 3 when δ drops to 50%. This system state corresponds to the case when the resource utilisation measures 75%.

[1] 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.

We are able to choose some combination of these metrics 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. [9]) which are able to determine *time of use* meters to reflect variations in price–time processing, and are the subject of future work.

6. Conclusions

Performance-based 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 how such services might be built based on existing Condor- and Globus-enabled Grid infrastructures.

A local (intra-domain) level predictive co-scheduler is described, which 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 makepsan 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 middleware 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.

References

[1] V. Adve, R. Bagrodia, J. Browne, E. Deelman et al., Poems: End-to-end performance design of large parallel adaptive computional systems, IEEE Transactions on Software Engineering 26 (11) (2000) 1027–1048.

[2] A. Alkindi, D. Kerbyson, G. Nudd, Optimisation of application execution on dynamic systems, Future Generation Computer Systems 17 (8) (2001) 941–949.

[3] R. Allan, J. Brooke, F. Costen, M. Westhead, Grid-based high performance computing, Technical Report of the UKHEC Collaboration UKHEC (2000), 2000.

[4] J. Aman, C. Eilert, D. Emmes, P. Yacom, D. Dillenberger, Adaptive algorithms for managing a distributed data processing workload, IBM Systems Journal 36 (2) (1997) 242–283.

[5] D. Arnold, J. Dongarra, The NetSolve environment: processing towards the seamless Grid, in: 29th Int. Conference on Parallel Processing, ICPP-2000, 21–24 August 2000, Toronto, Canada, 2000.

[6] 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, R. Wolski, The GrADS project: Software support for high-level grid application development, International Journal of High Performance Computing Applications 15 (4) (2001) 327–344.

[7] F. Berman, R. Wolski, S. Figueira, J. Schopf, G. Shao, Application-level scheduling on distributed heterogeneous networks, in: Proc. of Supercomputing, 1996.

[8] R. Buyya, D. Abramson, J. Giddy, Nimrod–G resource broker for service-oriented grid computing, IEEE Distributed Systems Online 2 (7) (2001).

[9] R. Buyya, D. Abramson, J. Giddy, H. Stockinger, Economic models for resource management and scheduling in Grid computing, Concurrency and Computation: Practice and Experience 14 (2002) 1507–1542.

[10] J. Cao, S. Jarvis, S. Saini, D. Kerbyson, G. Nudd, ARMS: An agent-based resource management system for grid computing, Scientific Programming 10 (2) (2002) 135–148.

[11] J. Cao, D. Kerbyson, G. Nudd, High performance service discovery in large-scale multi-agent and mobile-agent systems, International Journal of Software Engineering and Knowledge Engineering 11 (5) (2001) 621–641.

[12] J. Cao, D. Kerbyson, E. Papaefstathiou, G. Nudd, Performance modelling of parallel and distributed computing using PACE, in: 19th IEEE Int. Performance, Computing and Communication Conference, IPCCC'00, 2000, pp. 485–492.

[13] H. Casanova, J. Dongarra, NetSolve: A network server for solving computational science problems, International Journal of Supercomputer Applications and High Performance Computing 11 (3) (1997) 212–223.

[14] K. Czajkowski, GT resource management and scheduling: The globus perspective, in: GlobusWorld 2003, 13–17 January 2003, San Diego, CA, USA, 2003.

[15] P. Dinda, Online prediction of the running time of tasks, Cluster Computing 5 (3) (2002) 225–236.

[16] S. Fitzgerald, I. Foster, C. Kesselman, Grid information services for distributed resource sharing, in: 10th Int. Symposium on High Performance Distributed Computing, HPDC-10 '01, 2001.

[17] I. Foster, C. Kesselman, Globus: A metacomputing infrastructure toolkit, International Journal of Supercomputer Applications 11 (2) (1997) 115–128.

[18] I. Foster, C. Kesselman, The GRID: Blueprint for a New Computing Infrastructure, Morgan-Kaufmann, 1998.

[19] I. Foster, C. Kesselman, J. Nick, S. Tuecke, Grid services for distributed system integration, IEEE Computer 35 (6) (2002) 37–46.

[20] I. Foster, A. Roy, V. Sander, A quality of service architecture that combines resource reservation and application adaption, in: Proceedings of 8th Int. Workshop on Quality of Service, 5–7 June 2000, Pittsburgh, PA, USA, 2000, pp. 181–188.

[21] J. Frey, T. Tannenbaum, I. Foster, M. Livny, S. Tuecke, Condor-G: A computation management agent for multi-institutional grids, in: Proc. of the 10th IEEE Sym. on High Performance Distributed Computing, 2001.

[22] X. He, X. Sun, G. Laszewski, A QoS guided scheduling algorithm for grid computing, in: Int. Workshop on Grid and Cooperative Computing, GCC02, Hainan, China, 2002.

[23] C. Herder, J. Dujmovic, Frequency analysis and timing of Java bytecodes, San Francisco State University Technical Report, SFSU-CS-TR-00.02, 2000.

[24] D. Kerbyson, H. Alme, A. Hoisie, F. Petrini, H. Wasserman, M. Gittings, Predictive performance and scalability modelling of a large-scale application, in: Supercomputing'01, 2001.

[25] W. Leinberger, V. Kumar, Information power grid: The new frontier in parallel computing? IEEE Concurrency 7 (4) (1999).

[26] H. Lim Choi Keung, J. Dyson, S. Jarvis, G. Nudd, Performance evaluation of a grid resource monitoring and discovery service, IEE Proceedings-Software 150 (4) (2003).

[27] M. Litzkow, M. Livny, M. Mutka, Condor — a hunter of idle workstations, in: 8th International Conference on Distributed Computing Systems, ICDCS'88, 1988, pp. 104–111.

[28] W. McColl, Foundations of time-critical computing, in: 15th IFIP World Computer Congress, Vienna and Budapest, 1998.
[29] G. Nudd, D. Kerbyson, E. Papaefstathiou, S. Perry, J. Harper, D. Wilcox, PACE: A toolset for the performance prediction of parallel and distributed systems, International Journal of High Performance Computing Applications 14 (3) (2000) 228–251.
[30] S. Perry, R. Grimwood, D. Kerbyson, E. Papaefstathiou, G. Nudd, Performance optimisation of financial option calculations, Parallel Computing 26 (5) (2000) 623–639.
[31] R. Raman, M. Livny, M. Solomon, Matchmaking: Distributed resource management for high throughput computing, in: 7th Int. Symposium on High Performance Distributed Computing, HPDC-7'98, 1998.
[32] D. Spooner, S. Jarvis, J. Cao, S. Saini, G. Nudd, Local grid scheduling techniques using performance prediction, IEE Proceedings-Computers and Digital Techniques 150 (2) (2003) 87–96.
[33] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, U. Nagashima, Overview of a performance evaluation system for global computing scheduling algorithms, in: Proc. of 8th IEEE Int. Symp. on High Performance Distributed Computing, 1999, pp. 97–104.
[34] J. Turner, D. Spooner, J. Cao, S. Jarvis, D. Dillenberger, G. Nudd, A transaction definition language for Java application response measurement, Journal of Computer Resource Measurement 105 (2001) 55–65.
[35] R. Wolski, N. Spring, J. Haye, The network weather service: A distributed resource performance forecasting service for metacomputing, Future Generation Computer Systems 5–6 (1999) 757–768.

Stephen A. Jarvis is a senior lecturer in the High Performance Systems Group at the University of Warwick. He has authored more than 90 refereed publications (including three books) in the area of software and performance evaluation. While previously at the Oxford University Computing Laboratory, he worked on the development of performance tools with Oxford Parallel, Sychron Ltd., and Microsoft Research in Cambridge. He has close research ties with IBM, including current projects with IBM T. J. Watson Research Centre in New York and with IBM Hursley Park in the UK. Additional collaborative projects include those with BT at Martlesham Heath, Hewlett Packard Research Laboratories in Bristol and Palo Alto, and with the UK National Business to Business Centre. Dr Jarvis sits on a number of international program committees for high-performance computing, he is manager of the Midlands e-Science Technical Forum on Grid Technologies, and elected member of the EPSRC Review College.

Daniel P. Spooner is a lecturer in the Department of Computer Science at the University of Warwick and member of the High Performance Systems Group. His primary research interests are in the application of prediction techniques to improve resource scheduling in grid environments. Other interests include distributed network management architectures and high performance web services.

Hélène N. Lim Choi Keung is in the latter stages of her Ph.D. studies at the Department of Computer Science, University of Warwick. She received a B.Sc. in Computing Science from Aston University, Birmingham, and also worked as an e-Business Research Engineer at the International Automotive Research Centre, Warwick Manufacturing Group. Her current research interests are in distributed and Grid computing, with an emphasis on resource monitoring, selection, autonomous systems, and performance evaluation and prediction.

Junwei Cao is currently a Research Scientist at the Center for Space Research, Massachusetts Institute of Technology, USA. He has been working on grid infrastructure implementation and grid-enabled application development since 1999. Before joining MIT in 2004, Dr. Cao was a research staff member of C&C Research Laboratories, NEC Europe Ltd., Germany. He received his Ph.D. in Computer Science in 2001 from the University of Warwick, UK and M.Sc. in 1998 from Tsinghua University, China, respectively. He is a member of the IEEE Computer Society and the ACM.

Subhash Saini received his Ph.D. from the University of Southern California and has held positions at University of California at Los Angeles (UCLA), University of California at Berkeley (UCB), and Lawrence Livermore National Laboratory (LLNL). He has 11 years experience of teaching Physics at B.S. and M.S. level. Since 1989, he is a senior scientist at the NASA Advanced Supercomputing (NAS) program at NASA Ames Research Center. He is a senior visiting scientist at LLNL under a participating guest program. He has been a highly rated tutorial speaker at SC 92, SC '94, SC '95, SC '96, SC '97 and SC '98. His SC '94, SC '95, SC '96, SC '97 and SC '2004 tutorials on high end computing drew the highest number of attendees in any of the pre-conference tutorials. His research interests involves performance evaluation and modelling of new generation of CMOS based processors and highly parallel computers including next generation of petaflop class computers. He has published 127 technical papers and presented over 200 technical talks. He has won several awards for Excellence in Teaching including one from USC. In 1992, he was named the NAS employee of the year. In 2001, he was a co-author of a Best Technical Paper Award at SC 2001.

Graham R. Nudd is Head of the High Performance Computing Group and Chairman of the Computer Science Department at the University of Warwick. His primary research interests are in the management and application of distributed computing. Prior to joining Warwick in 1984, he was employed at the Hughes Research Laboratories in Malibu, California.