Performance-based middleware for Grid computing

G. R. Nudd and S. A. Jarvis*

High Performance Systems Group, Department of Computer Science,
University of Warwick, Coventry CV4 7AL, U.K.

SUMMARY

This paper describes a stateful service-oriented middleware infrastructure for the management of scientific tasks running on multi-domain heterogeneous distributed architectures. Allocating scientific workload across multiple administrative boundaries is a key issue in Grid computing and as a result a number of supporting services including match-making, scheduling and staging have been developed. Each of these services allows the scientist to utilise the available resources, although a sustainable level of service in such shared environments cannot always be guaranteed.

A performance-based middleware infrastructure is described in which prediction data for each scientific task is calculated, stored and published through a Globus-based performance information service. Distributing this data allows additional performance-based middleware services to be built, two of which are described in this paper: an intra-domain predictive co-scheduler and a multi-domain workload steering system. These additional facilities significantly improve the ability of the system to meet task deadlines, as well as enhancing inter-domain load balancing and system-wide resource utilisation.

KEY WORDS: Grid computing; middleware; performance prediction; scheduling; quality of service

1. INTRODUCTION

The use of performance-based management services is becoming increasingly common in Grid computing architectures [1, 2], and the deployment of these and related services [3, 4, 5, 6] is likely to increase as Grid computing evolves from an experimental scientific framework to a common customer-supporting computing platform.

*Correspondence to: Department of Computer Science, University of Warwick, Coventry CV4 7AL, U.K.
†E-mail: Stephen.Jarvis@warwick.ac.uk

Contract/grant sponsor: NASA AMES Research Center, EPSRC, EPSRC e-Science Core Programme; contract/grant number: N68571-01-C-0012, GR/R47424/01, GR/S03056/01

Copyright © 2000 John Wiley & Sons, Ltd.
Grid performance-services research is motivated by a number of different needs. The classification of these needs is approximate to the different levels of abstraction that comprise the architecture. At a systems level there is a requirement to make best use of the underlying architectural components, so as to justify investment and maximise the potential of the contributing parts (computational resources, network, data storage facilities etc). In analysing how to maximise resource potential, it is also possible to predict future demands and also calculate the impact of upgrading or reconfiguring the architecture at a later date. At the service level, vendors are obliged to ensure and maintain a stable provision of resource, while maximising throughput and reserving spare capacity for future real-time service demands. It is desirable to be able to provide different customers with different levels of service, and as such there needs to be a means of capturing different customer requirements and then mapping these requirements to service provision. At the end-user level there needs to be some degree of performance transparency and predictability in the way that supported applications operate. If the user is to pay for a level of service then this needs to be guaranteed; if it is possible to select alternative (slower, cheaper etc) services then this capability also needs to be realised in the supporting middleware utilities.

This paper extends the research documented in [7], describing a performance-based middleware infrastructure for the management of closely-coupled scientific tasks running on multi-domain heterogeneous distributed architectures. The middleware is based on the Performance Analysis and Characterisation Environment (PACE) [8] developed by the High Performance Systems Group at the University of Warwick. PACE is a state-of-the-art performance prediction system that provides quantitative data concerning the performance of scientific applications running on high performance parallel and distributed computing systems. The system works by modelling both the applications and the underlying hardware on which the applications are to be run, and combining the resulting models to derive predictive execution data. 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 [9].

The middleware is constructed in such a way that the performance data provided by PACE is published through a performance information service using the multi-domain management features of the Globus Toolkit [10, 11]. This ensures that the predictive data remains consistent amongst the distributed nodes in the system and that updates to the performance data are available throughout the system. The delivery of the data via information services allows it to be used for domain-level predictive resource scheduling and higher-level management of multi-domain grid system resources. A key feature of the implementation is that the prediction-enabled scheduling is implemented as an add-on feature to the intra-domain resource management system Condor [12].

The PACE toolkit has been used extensively to schedule and manage high-throughput scientific applications [8, 9, 13, 14, 15]. These applications are deadline driven and can be executed on any of the component resources, although cross-domain execution is generally avoided because of the latency and speed of the interconnection network. The operation of the supporting middleware is to best site the scientific task; this must be done in the context of the other tasks in the shared system and in response to a deadline assigned to that task. This operation is demonstrated using a 256-node grid, consisting of a heterogeneous collection
of resource domains each containing a homogeneous cluster or multiprocessor machine. This research differs from traditional distributed computing resource allocation and scheduling; the emphasis is on performance-aware multi-domain task management, where each domain is unaware of the resources offered by another, and the underlying architecture is designed to be large-scale and consists of non-dedicated resources whose computational capabilities vary over time.

The implementation, real-time capabilities and parametric prediction functions of PACE are described in Section 2. In Section 3 the integration of PACE with a predictive co-scheduler is documented; experimental data is included that demonstrates the added-value that prediction services contribute to intra-domain resource scheduling of closely coupled and deadline driven scientific tasks. The implementation of a multi-domain performance management system is detailed in Section 4; this includes the characteristics of a Globus-based performance information service and the supporting agent system that is used to advertise and discover resources based on this data. A case study is documented in Section 5. The results show that employing performance services at the intra- and multi-domain levels in a grid system provides an efficient framework for the management and distribution of scientific tasks in a large-scale, heterogeneous distributed computing environment.

2. THE PACE TOOLKIT

Details of the PACE toolkit can be seen in Figure 1. An important feature of the design is that the application and resource modelling is separated and that 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 process is automated by compiling the performance specification language (PSL) scripts into an application model. 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 rely on analytical models and micro-benchmarks (such as the CPU, network and memory) to determine the performance of the hardware platform on which the application is to be executed. The HMCL scripts provide a resource model for each hardware component in the system and 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 built, 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.
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) high performance demonstrator applications [9, 14]. 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. The success of PACE means that it has been used in a number of other high-performance settings, these include the performance optimisation of financial applications [16], real-time performance analysis and application steering [13] and the predictive performance and scalability modelling of the application Sweep3D [9].

3. INTRA-DOMAIN RESOURCE MANAGEMENT

The anticipated size of Grid computing environments necessitates a scalable approach to resource management. In order to harness the capabilities of established scheduling systems, there is a need for management frameworks that can consolidate key resource information from local-level schedulers and coordinate higher-level task submissions between managers. This requirement has directed a number of research projects including Condor-G for the access and remote management of Condor [12, 17] clusters, and Nimrod/G, for the Grid-enablement of Nimrod managed networks of workstations [18].

A similar two tiered design is used in this research. An agent system (documented in Section 4) provides multi-domain management to a Grid composed of a number of intra-domain
resources, which are themselves managed by a predictive co-scheduler known as Titan [15]. This separation is important as it emphasises the difference between intra-domain (local) resource management and multi-domain (wide-area) task management.

A unique feature of this work is that both intra-domain resource management and multi-domain task management are driven by performance prediction data. The granularity and application of this data is different at each level in the system, but the decision making at each level is supported by data supplied by performance information services (see Section 4).

The management of resources at the intra-domain level is provided by the combination of a scheduler coordinator (Titan [15]) and a standard commodity scheduler (in this case Condor [12], where Condor is operated in dedicated mode rather than the cycle-stealing mode). The Titan system uses 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 execution time of closely-coupled scientific applications by targeting suitable resources and scaling the application appropriately. This is not however performed in isolation as the system is also sensitive to surrounding tasks and requirements. At the core of the system is a genetic algorithm 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 system components.

Tasks enter the system by means of a portal (currently implemented as command-line utility) from which the user specifies the task 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 task to perform necessary initialisation, such as modifying input control files based on the processor mapping reserved by Titan, or downloading appropriate binaries for the resource type.

When a task request is received, Titan combines the application model with the hardware model for the local domain. PACE is used to evaluate this run-time scenario and explore different application/resource mappings to obtain a scaling graph for the task over the given set of resources. By comparing the task’s minimum run-time with the current status of the scheduler queue, Titan is able to predict when the task will complete and can compare this with the user-specified deadline. If the deadline of the task can be met, the task is submitted into the local domain for subsequent processing. If rejected, the system will negotiate with neighbouring schedulers to enquire whether the task request can be met elsewhere (see Section 4). If it is not possible to meet the deadline then the task is run on the resource that minimises the deadline failure.

When a task is accepted it is placed in Titan's scheduling queue with the other accepted tasks. Titans genetic algorithm then works on this queue while the jobs are waiting, exploring task mappings which reduce the makespan (end-to-end run-time), idle time (locked between processes) and the average delay (the amount of time tasks complete before or after their deadline). The genetic algorithm does 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 tasks.
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 (or indeed any other commodity scheduler) 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 \[4\] that are typically used to select and rank machines with specific configurations. In this work, this functionality is utilised by creating ClassAds that force Condor to run a task on designated machines using the name attribute. By keeping Condor’s queue empty and forcing the mapping of tasks, Titan is 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 small changes in state and compensate accordingly. 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. The capabilities of the predictive co-scheduling are demonstrated. This is done
Figure 3. (a) The chart at the top shows the run-time schedule queue for the 30 tasks using Condor. (b) The chart at the bottom shows how the Titan predictive co-scheduler improves this schedule for the same 30 tasks. The effect on makespan and idle time can be found in Table I.

by selecting 30 random tasks from a set of five 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 1.4GHz Pentium IVs with communication across a Fast Ethernet network. With a population of 40, the scheduler (running on an 800Mhz Pentium III) is capable of performing approximately
Table I. Experimental results comparing Condor (experiments 1 to 3) with Condor and the co-scheduler Titan (experiment 4). Run-time analysis of the first and last of these experiments is shown in Figure 3.

<table>
<thead>
<tr>
<th>CONFIGURATION</th>
<th>MAKESPAN (MIN)</th>
<th>IDLE (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Condor (with arbitrary number of hosts per task)</td>
<td>70.08</td>
<td>61</td>
</tr>
<tr>
<td>Condor (with maximum number of hosts per task)</td>
<td>69.10</td>
<td>28</td>
</tr>
<tr>
<td>Condor (with a pre-calculated number of hosts per task)</td>
<td>38.05</td>
<td>14</td>
</tr>
<tr>
<td>Condor and Titan</td>
<td>35.19</td>
<td>21</td>
</tr>
</tbody>
</table>

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 allowed.

The results of four experiments are shown in Table I. 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 task scales over the given resources. In many cases, larger tasks block smaller tasks (see Figure 3a) 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 processes may be marginal and blocking is still common.

In the third experiment the tasks are submitted with a pre-calculated number of hosts, resulting in a small run-time without over-utilising 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 they are mapped to the physical resources by Condor. The new schedule can be seen in Figure 3b.

The results in Table I demonstrate the significant improvement obtained using this prediction co-scheduling technique. Over the first two experiments the Condor-Titan system effectively halves the makespan (from 70 minutes to 35 minutes). 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 improvements are significant, but of additional interest is the ability of the predictive co-scheduling to manage task deadlines and thus integrate extra quality of service features. Condor has no facility to reorder tasks and therefore makes no provision for task deadlines. The impact of this facility is explored in detail in Section 5.
4. MULTI-DOMAIN TASK MANAGEMENT

Intra-domain application performance and resource usage information is stored and made available throughout the multi-domain system components via a performance information service. There is still some debate as to how this information services framework should be implemented in a highly distributed infrastructure (and how to avoid bottlenecks in information storage and retrieval, how to ensure consistency of information, what data models should be used etc) [19, 20, 21, 22, 23]. The approach used in this research is based on the Monitoring and Discovery Service (MDS) [10] from the Globus Toolkit [11]. 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 supported through a relational database, and this information is transferred to the GRIS host through LDIF (LDAP Data Interchange Format) supported information providers (see 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.

The data inside the performance information service is utilised by a network of agents (see Figure 5) which cooperate to provide multi-domain task management. The agents provide a higher-level approach to task management 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 brokerage as new demands are placed on the system. The network of agents is dynamic and so the representation of resources is flexible in so much as it allows resources to leave or join the grid at any time. The agent system itself is well documented [24, 25]; some detail is provided in order to understand the case study in Section 5.

Each agent is composed of a series of layers: communication - through which agents are able to communicate with each other using common data models and communication protocols, this also provides an interface to heterogeneous networks and operating systems; coordination - from which task (execution) requests are submitted directly to the agent either manually or through a job submission portal, an agent will then allocate the task to the local resources (if they are able to process the request) or initiate some higher-level resource discovery service; management - at which decision making is supported through an interface with the local resource manager and information service provider.

The agent system submits to and reads from the performance information service. The information supported by the agents is organised in a number of agent capability tables (ACTs). These are currently: the $T_{ACT}$ - service information of the resources which the agent represents; $L_{ACT}$ - information of the services found lower in the agent network; $G_{ACT}$ - service information from services found higher in the agent network. Some notion of a hierarchy is needed to support this organisation.
Figure 4. Implementation of a performance information service provided through the Globus MDS. Data is stored in a local relational database and data exchange is managed through a number of LDIF supported information providers.

Figure 5. The interaction of the agent hierarchy with the intra-domain schedulers. It is perfectly feasible to replace the intra-domain schedulers (in this case Titan) with other commodity systems.
The content of the ACTs is maintained through two methods of service update, these are data-pull - an agent makes a request for data, and data-push - service information is emitted asynchronously. The update of information can take place periodically or when data in the network changes. The agent system is configured so that it can invoke service advertisement and service discovery. When searching for services, each agent will first evaluate whether the request can be met locally (by querying its TACT); if this is not the case then the services provided by the neighbouring resources are queried (through the LACT and the GACT) and the request is dispatched to the agent which is able to provide the best service. If none of the agents are able to meet the request then it is sent to a higher-level node. The process of discovery terminates when the head of the logical hierarchy is reached.

There are many other ways of configuring this level of management (such as using economic models [26] for example). While this approach is not intended to deliver an optimally balanced Grid system, it does provide a robust system in which resources are located simply and efficiently and requests tend to migrate to local rather than global resources. The system also scales well as there is never a need to broadcast advertisement or discovery requests.

5. CASE STUDY: PREDICTION-BASED TASK MANAGEMENT

The experimentation in Section 3 is extended through simulation to a multi-domain heterogeneous collection of resources. This is configured as 16 resource domains (\(D_0\) to \(D_{15}\)), each containing 16 homogeneous processors/hosts. The resource capabilities of each of the domains is different, they include single multiprocessor machines or clusters of commodity workstations. In order, their computational capabilities are (high) \(D_0, D_5, D_1, D_3, D_9, D_{10}, D_{12}, D_2, D_4, D_8, D_{13}, D_0, D_{11}, D_{14}, D_{10}, D_{15}\) (low).

Each domain is represented by an agent, and each agent maintains service information through the ACTs. A number of experiments are run in which 200, 500 and 1000 application requests (\(r\)) were 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. The deadlines for each task were randomly selected from the range of predicted values, with suitable time allowed for network latency.

Four performance metrics are defined in order to allow the comparison between different intra- and multi-domain resource and task management configurations:

5.1. Total application execution time

The total application execution time (the makespan) is defined as the period of time \(t\) over which a set of \(m\) parallel tasks are scheduled on a set of resources:

\[
t = \max_{1 \leq j \leq m} \{te_j\} - \min_{1 \leq j \leq m} \{ts_j\}
\]

(1)

where \(te_j\) is the earliest predicted completion time for task \(j\), and \(ts_j\) is the earliest possible start time.
5.2. Average delay

The average delay \( (\epsilon) \) is the number of seconds that the tasks complete before (or after) their deadlines. The aim is that this value is positive, therefore indicating that more time is gained through tasks completing before their deadlines than lost through tasks failing to meet their deadlines:

\[
\epsilon = \frac{\sum_{j=1}^{m} (tr_j - te_j)}{m}
\]

where \( tr_j \) is the actual run time for task \( j \).

5.3. Average resource utilisation rate

The resource utilisation rate \( \nu_i \) for each host \( H_i \) is calculated as follows:

\[
\nu_i = \frac{\sum_{j,M=1}^{M} (te_j - ts_j)}{t} \times 100\%
\]

The average resource utilisation rate \( \nu \) for all of the hosts \( H \) is therefore:

\[
\nu = \frac{\sum_{i=1}^{n} \nu_i}{n}
\]

5.4. Load balancing level

The mean square deviation of \( \nu_i \) is defined as:

\[
d = \sqrt{\frac{\sum_{i=1}^{n} (\nu - \nu_i)^2}{n}}
\]

and the relative deviation of \( d \) over \( \nu \), that describes the level of load balancing in the system, is:

\[
\beta = \left( 1 - \frac{d}{\nu} \right) \times 100\%
\]

Load balancing is deemed most effective when \( d \) equals zero and \( \beta \) equals 100%.

The experimental results in Table II represent two scenarios:

- In the first case each task is executed in the local domain in a first-come-first-served order. No attempt is made to improve intra-domain resource scheduling (using Titan) or the multi-domain management of tasks (using the agent system) - that is, the predictive middleware (\( M \)) is inactive.
- In the second case the Titan co-scheduler is used at the intra-domain level and the agent system is employed for multi-domain task management - that is, the predictive middleware (\( M \)) is active.
Table II. 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; \( \epsilon \) is the deadline-based average delay; \( \nu \) is the resource utilisation rate and \( \beta \) the load balancing. The last two rows correspond to the run-time views shown in Figures 6 and 7.

<table>
<thead>
<tr>
<th>( r )</th>
<th>( r/s )</th>
<th>( M )</th>
<th>( t(s) )</th>
<th>( \epsilon(s) )</th>
<th>( \nu(%) )</th>
<th>( \beta(%) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>ON</td>
<td>OFF</td>
<td>839</td>
<td>-1</td>
<td>24</td>
<td>36</td>
</tr>
<tr>
<td>200</td>
<td>ON</td>
<td>ON</td>
<td>302</td>
<td>78</td>
<td>51</td>
<td>63</td>
</tr>
<tr>
<td>200</td>
<td>OFF</td>
<td>OFF</td>
<td>812</td>
<td>-36</td>
<td>25</td>
<td>37</td>
</tr>
<tr>
<td>200</td>
<td>ON</td>
<td>ON</td>
<td>245</td>
<td>72</td>
<td>50</td>
<td>67</td>
</tr>
<tr>
<td>200</td>
<td>OFF</td>
<td>OFF</td>
<td>814</td>
<td>-64</td>
<td>24</td>
<td>35</td>
</tr>
<tr>
<td>200</td>
<td>ON</td>
<td>ON</td>
<td>218</td>
<td>62</td>
<td>49</td>
<td>58</td>
</tr>
<tr>
<td>500</td>
<td>OFF</td>
<td>OFF</td>
<td>1784</td>
<td>-112</td>
<td>28</td>
<td>51</td>
</tr>
<tr>
<td>500</td>
<td>ON</td>
<td>ON</td>
<td>633</td>
<td>78</td>
<td>57</td>
<td>69</td>
</tr>
<tr>
<td>500</td>
<td>OFF</td>
<td>OFF</td>
<td>1752</td>
<td>-205</td>
<td>29</td>
<td>53</td>
</tr>
<tr>
<td>500</td>
<td>ON</td>
<td>ON</td>
<td>461</td>
<td>66</td>
<td>57</td>
<td>80</td>
</tr>
<tr>
<td>500</td>
<td>OFF</td>
<td>OFF</td>
<td>1877</td>
<td>-276</td>
<td>27</td>
<td>51</td>
</tr>
<tr>
<td>500</td>
<td>ON</td>
<td>ON</td>
<td>305</td>
<td>32</td>
<td>68</td>
<td>79</td>
</tr>
<tr>
<td>1000</td>
<td>OFF</td>
<td>OFF</td>
<td>2752</td>
<td>-314</td>
<td>36</td>
<td>59</td>
</tr>
<tr>
<td>1000</td>
<td>ON</td>
<td>ON</td>
<td>1160</td>
<td>79</td>
<td>61</td>
<td>84</td>
</tr>
<tr>
<td>1000</td>
<td>OFF</td>
<td>OFF</td>
<td>2606</td>
<td>-493</td>
<td>39</td>
<td>59</td>
</tr>
<tr>
<td>1000</td>
<td>ON</td>
<td>ON</td>
<td>669</td>
<td>65</td>
<td>74</td>
<td>89</td>
</tr>
<tr>
<td>1000</td>
<td>OFF</td>
<td>OFF</td>
<td>2756</td>
<td>-681</td>
<td>36</td>
<td>58</td>
</tr>
<tr>
<td>1000</td>
<td>ON</td>
<td>ON</td>
<td>467</td>
<td>-6</td>
<td>77</td>
<td>84</td>
</tr>
</tbody>
</table>

In the case when the system load and submission rate is low (200 requests submitted 1 per second) the first-come-first-served implementation is able to meet most of the deadlines (the average delay \( \epsilon \) is -1). As the submission rate increases, so the ability to meet these deadlines decreases (\( \epsilon \) increases to -36 at a submission rate of 2 requests per second and to -64 at 5 requests per second); this trend is also demonstrated at the higher workloads.

Activating the intra-domain and multi-domain predictive management has a positive effect on \( \epsilon \); when 200 requests are sent 1 per second, \( \epsilon \) equals 78, 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 under schedule.

The improvements to the makespan \( t \) and average delay \( \epsilon \) are made through global and local level optimisations. This can be observed through the metrics for resource utilisation \( (\nu) \) and load balancing \( (\beta) \). In the case when the workload and submission rates are high, the first-come-first-served implementation achieves a system-level balance of 58%, while the resource utilisation rate is 36%. There is a significant improvement to the values of these metrics when the predictive middleware is activated, the system balance increases to 84% and the utilisation
rate to 77%. The result of this has a large impact on the overall makespan which is improved by 83%. The detail as to how these improvements are made is observed through the analysis of this high-workload case.

A domain-level breakdown of these performance metrics is presented below. Three experimental cases are included, the results of which can be found in Table III:

- The case when the predictive middleware is inactive, the OFF case above;
- An intermediate case where the agent system is activated so that multi-domain predictive management is performed, but intra-domain predictive co-scheduling (using the Titan system) remains inactive;
- The case when both intra-domain predictive co-scheduling and multi-domain level task management are used, the ON case above.

The high request and submission rate imposes a large workload on each of the contributing domains. The domains that contain the more powerful resources, $D_0$ and $D_7$, are better equipped to meet the deadlines for the tasks they receive than their less powerful counterparts, $D_6$ and $D_{15}$; this is reflected in $e$. The resource utilisation rate $\nu$ of the more powerful resources is also low (12 and 14%, as opposed to 54 and 51%). These results can be seen in the run-time view in Figure 6.

![Table and Figure]

Figure 6. The results of running 1000 tasks submitted at a request rate of 5 per second. This corresponds to the first scenario, when the predictive middleware ($M$) is turned OFF. Note that the makespan is 2756 seconds.
Enabling the agent system (in the intermediate case) increases the number of requests directed to these more powerful resources ($\nu$ of $D_0$ and $D_7$ increases to 55 and 46%), an effect which improves the overall system balance $\beta$ to 81%. This improvement is also reflected in the new makespan ($t$) of 1659 seconds (a reduction of 40% over the case when the middleware is off).

Considerable further improvements can be made by employing predictive co-scheduling at the intra-domain level (when all the middleware is on). The resource utilisation rate $\nu$ of all resources improves to 77%, a marginal improvement is made to the system balance, yet there is a significant reduction in $t$ to 467 seconds (an improvement of 83% over the case when the middleware is off). These results can be seen in the run-time view in Figure 7.

The use of predictive data for multi-domain task management does increase system balance, particularly when the load and submission rate is high. This mechanism will distribute tasks to more powerful global resources, although the overall resource utilisation may remain low. In order to improve resource utilisation (as well as load balancing), additional intra-domain level management is needed.
Table III. Domain breakdown of experimental results for 1000 requests at 5 per second: When the predictive middleware (at the intra- and multi-domain level) is off; an intermediate case when multi-domain task management is performed but intra-domain resource management is not; when both levels of predictive middleware are activated.

<table>
<thead>
<tr>
<th></th>
<th>Middleware OFF</th>
<th>Intermediate</th>
<th>Middleware ON</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_0$</td>
<td>-115</td>
<td>12</td>
<td>-611</td>
</tr>
<tr>
<td>$D_1$</td>
<td>-481</td>
<td>27</td>
<td>-567</td>
</tr>
<tr>
<td>$D_2$</td>
<td>-710</td>
<td>37</td>
<td>-575</td>
</tr>
<tr>
<td>$D_3$</td>
<td>-320</td>
<td>22</td>
<td>-309</td>
</tr>
<tr>
<td>$D_4$</td>
<td>-766</td>
<td>40</td>
<td>-538</td>
</tr>
<tr>
<td>$D_5$</td>
<td>-1171</td>
<td>54</td>
<td>-577</td>
</tr>
<tr>
<td>$D_6$</td>
<td>-1161</td>
<td>54</td>
<td>-544</td>
</tr>
<tr>
<td>$D_7$</td>
<td>-137</td>
<td>14</td>
<td>-377</td>
</tr>
<tr>
<td>$D_8$</td>
<td>-533</td>
<td>33</td>
<td>-437</td>
</tr>
<tr>
<td>$D_9$</td>
<td>-581</td>
<td>27</td>
<td>-540</td>
</tr>
<tr>
<td>$D_{10}$</td>
<td>-351</td>
<td>24</td>
<td>-322</td>
</tr>
<tr>
<td>$D_{11}$</td>
<td>-996</td>
<td>53</td>
<td>-544</td>
</tr>
<tr>
<td>$D_{12}$</td>
<td>-624</td>
<td>35</td>
<td>-598</td>
</tr>
<tr>
<td>$D_{13}$</td>
<td>-946</td>
<td>49</td>
<td>-409</td>
</tr>
<tr>
<td>$D_{14}$</td>
<td>-976</td>
<td>46</td>
<td>-418</td>
</tr>
<tr>
<td>$D_{15}$</td>
<td>-1097</td>
<td>51</td>
<td>-580</td>
</tr>
<tr>
<td><strong>System</strong></td>
<td><strong>-681</strong></td>
<td><strong>36</strong></td>
<td><strong>-502</strong></td>
</tr>
</tbody>
</table>

5.5. Assessing the quality of service

Quality of service is represented in these experiments as the ability of the system to satisfy the execution deadlines of all the tasks submitted. Although this analysis is performed on a per-task basis, it is equally feasible to group tasks to classes or priority of user, the net effect would be the same.

Figure 8 shows the average delay $\epsilon$ (in seconds) for each of the 16 domains in the experimental system. The results are presented for the light workload (200 tasks submitted at a rate of 1 per second, graph (a)) and for the heavy workload (1000 tasks submitted at a rate of 5 per second, graph (b)) and represent the case when the predictive middleware is either switched on or off; the figure therefore shows the effect on $\epsilon$ of the middleware services for a light and heavily loaded distributed system. It is noted that the medium workload displays similar characteristics between these two.

The results for the average delay are significant in a number of ways:
• They demonstrate that as the workload increases, the domains with the lowest computational capacity are less able to maintain $\epsilon$; see the OFF case in domains 5, 6, 11, 14 and 15, particularly in Figure 8b.
• They also show that when the middleware is enabled, it draws $\epsilon$ towards 0, that is, where the deadlines are maintained; see the ON case in domains 5, 6, 11, 14 and 15. It is also significant that it does so in a systematic way so that $\epsilon$ is maintained across all domains in the system (demonstrated by the near flat line in Figures 8a and 8b).
• The ability of the middleware to maintain a system-wide balance of $\epsilon$ is illustrated across different workloads. However, as the load on the system increases so the system-wide value of $\epsilon$ moves towards (and then below) 0.

Some of these results are to be expected, for example one would expect the ability of the poorer capacity resources to meet $\epsilon$ to decrease as the load increases. However, it is significant that the middleware is able to improve $\epsilon$ at both the intra- and multi-domain levels and that this is maintained through varying loads, even as the capacity of the system as a whole is reached.

5.6. Assessing the resource utilisation

The effect of the middleware on resource utilisation is also investigated. The difference in resource utilisation, between when the middleware is on and off, can be calculated (see Figure 9).

The difference in resource utilisation for the domains with the lowest computational capabilities is small (it is at its lowest point in domains 6 and 15); the difference is large in the domains with the highest computational capabilities (it is at its highest point in domains 0, 1, 7, 9). These results are caused by the predictive middleware being able to identify and make use of the large proportion of idle time on the higher capacity resources; even when the load is low, the percentage difference in $D_0$ (one of the domains containing the more powerful resources) is $40\%$, the percentage difference for $D_0$ (one of the domains containing the less powerful resources) is $4\%$. This trend is uniform across the different workloads and simply reflects the fact that the percentage resource utilisation increases as the overall workload increases.

The overall resource utilisation $\nu$ for each of the experimental cases can be found in Table III.

The impact of the middleware on the measures of quality of service and resource utilisation is significant. Further experimentation is proposed to extend the definitions of these metrics and also to explore the effect of the middleware on wider classes of application.

6. CONCLUSIONS

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 performance services might be built for such systems using Condor- and Globus-based infrastructure.
Figure 8. The average delay $\epsilon$ (in seconds) for each domain in the experimental Grid. The light workload (a) is shown at the top and the heavy workload (b) is shown at the bottom. The results represent when the predictive middleware is both ON and OFF.
Performance prediction data is generated by the PACE toolkit and is made available through an information service based on the Globus MDS. This data can then be used by supporting performance services, both at the intra-domain level (demonstrated through the prediction-based Titan co-scheduler) and at the level of multi-domain task management (demonstrated through the supporting agent system).

The impact of this approach is explored through four performance cost metrics: makespan, average delay, resource utilisation and load balance. It is found that the use of predictive data for multi-domain task management does increase system balance, particularly when the load and submission rate are high. What is striking, however, is that moving tasks to more powerful global resources does not necessarily improve overall resource utilisation and this must be achieved through additional intra-domain-level management.

A multi-tiered approach to service management also significantly improves the ability of the system to meet task deadlines. Using this form of management, the average delay is improved at both the intra-domain and multi-domain levels, and this is maintained through varying system loads until the system reaches capacity.

There are a number of areas in which this study can be extended. It should be feasible to substitute the performance data derived from the PACE toolkit with other forms of data, for example that derived from monitoring. Extending the flexibility of the middleware to include data from other sources (e.g., [27]) is the subject of future work.
Throughout these experiments the degree of predictive accuracy remained constant. In real systems it is likely that the performance data on the basis of which scheduling or resource allocation decisions are made is highly variable, inaccurate or indeed unavailable. This topic has received some attention [28, 29] and is the subject of future work [30].

ACKNOWLEDGEMENTS

The authors are grateful for the contributions made by Junwei Cao, Justin Dyson, Darren Kerbyson, Helene Lim Choi Keung, Daniel Spooner and Lei Zhao.

REFERENCES


