Performance Prediction and its use in Parallel and Distributed Computing Systems

Stephen A. Jarvis, Daniel P. Spooner, Helene N. Lim Choi Keung, Graham R. Nudd
High Performance Systems Group, University of Warwick, Coventry, UK
saj@dcs.warwick.ac.uk

Junwei Cao
C & C Research Laboratories
NEC Europe Ltd., Sankt Augustin, Germany

Subhash Saini
NASA Ames Research Center
Moffett Field, California, USA

Abstract

A performance prediction framework is described in which predictive data generated by the PACE toolkit is stored and published through a Globus MDS-based performance information service. Distributing this data allows additional performance-based middleware tools to be built; this paper describes two such tools, a local-level scheduler and a system for wide-area task management. Experimental evidence shows that by integrating these performance tools for local- and wide-area management, considerable improvements can be made to task scheduling, resource utilisation and load balancing on heterogeneous distributed computing systems.

1. Introduction

The computing architectural landscape is changing. Resource pools 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 [20] and the grid computing [13, 18] paradigm, aim to create integrated computational and collaborative environments that provide technology and infrastructure support for the efficient use of remote high-end computing platforms. The success 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, particularly if they are to become the computing platforms of choice.

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 topic 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 and the answers to each 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 and computational resources, for example the communication latencies provided by network monitors such as NWS [26].

How is this performance data obtained? Gathering performance data can be achieved by 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 [10]. 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, 21] – 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 [25] or different machine instruction benchmarks [16] 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 [4, 24], batch queueing [19], resource discovery [7, 8] or resource brokerage [6, 15]. 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 [14] 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) [21] developed by the High Performance Systems Group at the University of Warwick is a state-of-the-art 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 [9].

The real-time capabilities and parametric prediction functions (discussed in section 2) allow PACE to be used for the provision of dynamic performance information services (see section 3). These in turn can be used to aid the scheduling of tasks over clusters of homogeneous resources (see section 4), and provide a basis for the higher-level management of grid system resources (see section 5). Results show that employing performance 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 Figure 1. An important feature of the design is that the application and resource modelling is 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.

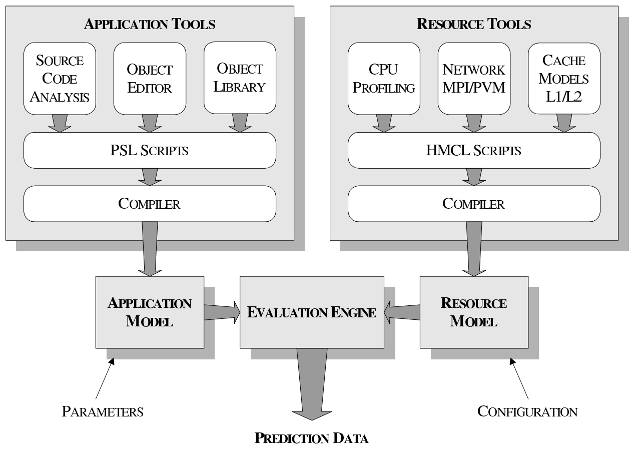

Figure 1. An outline of the PACE system including the application and resource modelling components and the parametric evaluation engine which combines the two.

The 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 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.

The PACE performance evaluation and prediction capabilities have been validated using ASCI (Accelerated Strategic Computing Initiative) high performance demonstrator applications [9, 17]. 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 [22], real-time performance analysis and application steering [2] and the predictive performance and scalability modelling of the ASCI application Sweep3D [9].

3. Performance Information Services

Once the prediction data has been generated by the PACE toolkit, it needs to be stored and made available to the system components via performance information services. There is still some debate as to how best to implement an information services framework 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 – see [10]). The approach used in this research is based on the Monitoring and Discovery Service (MDS) [11] from the Globus toolkit [12]. This consists of a number of configurable information providers (Grid Resource Information Services) and configurable directory components (Grid Index Information Services).

Information about each local collection of resources (a parallel machine or cluster of workstations, etc) 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 2).

Each GRIS host provides a subset of information about resources participating in the grid. Higher-level (global) access to the information in each of these local subsets 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 performance information service can work in two ways; as well as storing and distributing raw performance data from the PACE toolkit, additional *meta-data*[1] can be

[1]which in this context means data which is created from the mathematical analysis of the simpler performance data parts.

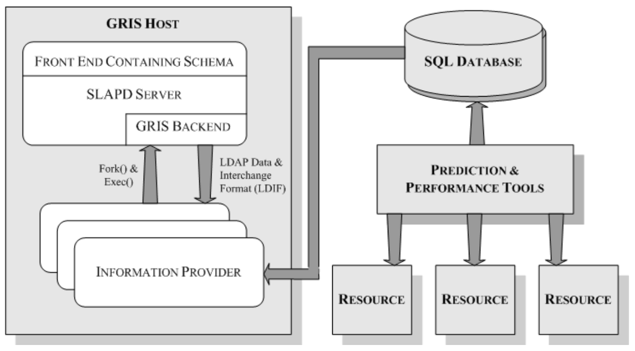

Figure 2. Implementation of a performance information service provided through the Globus MDS. Data is stored in a local relational database and data exchange is provided through a number of LDIF supported information providers.

generated and stored. One way in which this is achieved is through the use of a prediction-driven scheduler.

Using the predictive data for each application it is possible to estimate how those applications might map onto a set of resources. At the very least this provides a means for predicting the length[2] of the task queue for a particular set of resources. This meta-data is also stored and accessed through the performance information service.

Examples of this data, and the service provision which it allows, are given in sections 4 and 5. Supporting results demonstrate the effect these performance services have on task management and resource utilisation.

4. Scheduling over Local Resources

The anticipated size of grid computing environments necessitates a scalable approach to resource management. In order to harness the capabilities of established scheduling systems, it is beneficial to employ management frameworks that can both consolidate key resource information from local-level schedulers and also coordinate higher-level task submissions between managers. This requirement has directed a number of research projects including Condor-G, for the management of Condor [15, 19] clusters, and Nimrod-G, for the management of Nimrod [5, 6] experiments with grid protocols.

A similar two tiered design is used in this research. An agent system (documented in section 5) provides high-level management to a grid resource composed of a number of local-level clusters, which are themselves managed by a cluster scheduler known as Titan [23]. This separation is important as it emphasises the difference between local scheduling and wider-area task management, the characteristics and requirements of each being quite different.

[2]in terms of time rather than number of tasks

A unique feature of this work is that both local-level scheduling and high-level 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 the performance information services. Both the scheduling service and also the agent system add meta-data to the information services as well as using raw (and meta-) data from it.

A characteristic of this management system is that when applications are submitted for execution, they are given a user (or middleware) defined deadline. The aim of the local scheduler is to ensure that the mapping of tasks to resources is done in such a way that the deadlines are met. The scheduler is able to experiment with different *runtime scenarios* which allows the impact that the task mapping has on the overall makespan (run-to-completion time) to be investigated.

4.1. Prediction-based scheduling

The Titan scheduler combines PACE model evaluation with an iterative (genetic) algorithm. Such algorithms are suitable to problems of this nature where multiple (and often conflicting) metrics must be considered and where incremental changes to the problem space will occur over time. The application of genetic algorithms to multi-processing systems is well documented and this approach has previously been applied to a number of similar areas such as dynamic load balancing.

PACE provides a means of dynamically obtaining runtime estimates for different applications on different resources through the performance information services framework. Assuming that a run-to-completion (non-overlap) strategy is used, Titan can estimate the completion time for independent tasks by summing the earliest possible start time with the predicted execution time. Additionally, PACE can provide scaling analysis, so that Titan can limit the number of processing elements assigned to a task depending on its scaling behaviour.

The object of the scheduler is to produce schedules that fulfil system and user requirements, by selecting suitable candidate schedules from an existing set and producing new modified schedules with a measurable performance gain. A fitness function is used to guide this selection process by retaining successful schedules in the representation space. Titan generates an initial set of schedules, evaluates the schedules to obtain a measure of fitness, selects the most appropriate and combines them using genetic operators (crossover and mutation) to formulate a new set of solutions. If all of the hosts that are mapped to a task at the front of the queue are free, then the task is executed and then removed from the queue.

The fitness function is derived from a weighted cost function that evaluates the schedule makespan, the percentage of processor-idle time and the ability of the schedule to meet the task deadlines. Each schedule in the solution set can be compared by its relative fitness to obtain a rank position, which is subsequently used to drive the selection.

The capabilities of the predictive scheduler are demonstrated with a workload of 64 tasks queued onto 16 hosts. The tasks are selected from a set of parallel kernels for which PACE models are available and which exhibit different scaling behaviours (see Figure 3).

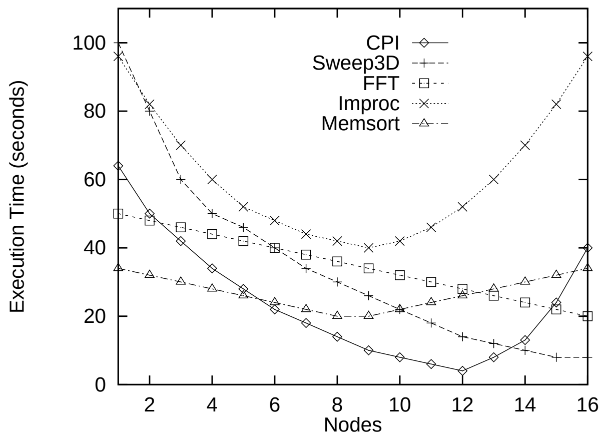

Figure 3. Set of example applications for which PACE performance models were available. The applications were chosen as they represent typical scientific computing programs that exhibit different parallel scaling characteristics.

The timings in Figure 3 are based on measurements obtained from a cluster of 800Mhz Pentium IIIs with communication timings measured across a Fast Ethernet network. With a population of 40, the scheduler (also running on an 800Mhz Pentium III) is capable of performing approximately 100 GA iterations per second. Each task is assigned an arbitrary deadline, all the tasks run to completion and pre-empting (the ability to multi-task or micro-schedule) is not allowed.

The results in Table 1 demonstrate the significant improvement (approximately 60% reduction in makespan) obtained using Titan as opposed to a simple batch queueing technique.

While batch queueing might be regarded as a worst-case solution, it is commonly used in cluster resource management systems; tasks are run in the order in which they arrive and additional priorities are the only means by which tasks can artificially move up the queue. The improvements which the Titan system is able to achieve are due (in part) to the ability to select and experiment with the predicted scalability of each task; PACE will restrict the node usage

	Makespan (t)		Advance (ε)	
Job	$Batch$ (s)	GA (s)	$Batch$ (s)	GA (s)
8	315	159	331	311
16	566	237	376	394
24	855	310	191	441
32	1186	431	92	394
40	1426	543	-62	364
48	1738	630	-190	341
56	1991	737	-333	292
64	2280	802	-473	261

Table 1. Improvement in t and ε of the Titan predictive scheduler over a simple batch queueing technique.

to that which is seen as most efficient within the context of the available hosts and the other tasks in the queue. Furthermore, Titan can also use PACE to sort tasks and change host mappings to reduce makespan. Efficient resource usage typically leads to a greater ability to meet deadlines (see ε in Table 1), although the GA also performs a light sort on the deadline through the weighted cost function.

These performance metrics are defined in order to allow the comparison of different scheduling and workload management configurations:

- *Total application execution time* (t) – the period of time when a set of parallel tasks m are scheduled onto a set of resources: $t = \overset{1 \leq j \leq m}{max}\{te_j\} - \overset{1 \leq j \leq m}{min}\{ts_j\}$, where te_j is the earliest possible completion time for task j, and ts_j is the earliest possible start time;

- *Average advance time* (ε) – the number of seconds that the tasks complete before their deadlines: calculated as $\varepsilon = \frac{\sum_{j=1}^{m}(tr_j - te_j)}{m}$, where tr_j is the actual run time for task j;

- *Resource utilisation rate* (ν_i) – which for each host H_i is: $\nu_i = \frac{\sum_{\forall j: i \mapsto j}(te_j - ts_j)}{t}$; the average resource utilisation rate ν for all n of H is then: $\nu = \frac{\sum_{i=1}^{n}\nu_i}{n}$;

- *Load balancing level* (β) – is the mean square deviation of ν_i, $d = \sqrt{\frac{\sum_{i=1}^{n}(\nu - \nu_i)^2}{n}}$, as a relative deviation over ν: $\beta = 1 - \frac{d}{v}$. The most effective load balancing is when d equals 0 and β equals 1.

5. Wide-area Task Management

Wide area task management is provided through a network of agents. These offer a less combinatorial approach to wide-area scheduling and as a result deliver increased scalability and adaptability. The agent system itself is well documented [7, 8]; some detail is provided in order to understand the case study in section 5.1.

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 service discovery; *management* – at which decision making is supported through an interface with the local resource manager or information service provider.

Each agent forms part of a high-level resource network, with each agent typically mapping to a set of resources, yet providing high-level coordination through a process of resource brokerage. This network is dynamic, so agents are able to join or leave at any time.

The agent system submits to and reads from the performance information services. 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.

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 T_{ACT}); if this is not the case then the services provided by the neighbouring resources are queried (through the L_{ACT} and the G_{ACT}) 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 (and a search for suitable resources is deemed to have failed).

There are many other ways of configuring this level of management. While this approach is not intended to deliver an optimally balanced grid system, it does provide a 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.

5.1. Prediction-based task management

r	r/s	M	t(s)	ε(s)	ν(%)	β(%)
200	1	OFF	354	-1	24	36
200	1	ON	241	78	51	63
200	2	OFF	377	-36	25	37
200	2	ON	169	72	50	67
200	5	OFF	332	-64	24	35
200	5	ON	188	62	49	58
500	1	OFF	1224	-112	28	51
500	1	ON	581	78	57	69
500	2	OFF	1123	-205	29	53
500	2	ON	349	66	57	80
500	5	OFF	1241	-276	27	51
500	5	ON	260	32	68	79
1000	1	OFF	2446	-314	36	59
1000	1	ON	1109	79	61	84
1000	2	OFF	2348	-493	39	59
1000	2	ON	646	65	74	89
1000	5	OFF	2545	-681	36	58
1000	5	ON	438	-6	77	84

Table 2. Experimental results: r is the total number of requests (load); r/s is the request submission rate per second; M represents whether predictive management is active; t is the makespan; ε is the deadline-based average advance time; ν is the resource utilisation rate and β the load balancing.

The experimentation in section 4.1 is extended to a network of resources consisting of 16 heterogeneous clusters (numbered C_0 to C_{15}), each containing 16 homogeneous processors/hosts. The resource capabilities of each of the clusters is different, ranging from SGI multiprocessors (C_0 and C_7) to clusters of SPARCstation 2s (C_6 and C_{15})[3].

Each cluster resource 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.

[3]C_1, C_3, C_9 and C_{10} represent clusters of 16 Sun Ultra 10s; C_{12} represents a cluster of 16 Sun Ultra 5s; C_2, C_4, C_8 and C_{13} represent clusters of 16 Sun Ultra 1s, and C_5, C_{11} and C_{14} represent clusters of 16 SPARCstation 5s.

The experimental results represent two scenarios:

- In the first case each of the local clusters executes the tasks it receives in a first-come-first-served order. No attempt is made to improve the local-level scheduling (using Titan) or the wide-area management of tasks (using the agent system) – that is, the predictive management system (M) is inactive.

- In the second case the Titan scheduler is used at the cluster level and the agent system is employed for wide-area task management – that is, the predictive management system (M) is active.

The results for these experiments are found in Table 2.

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 set (the average advance time ε is -1). As the submission rate increases, so the ability to meet these deadlines is diminished (ε 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 local- and global-level predictive management has a positive effect on ε; when 200 requests are sent 1 per second, ε 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), the prediction-based middleware is able to reduce this to -6 seconds.

The improvements to the makespan t and average advance time ε are made through global and local level optimisations. This can be observed through the metrics for resource utilisation (ν) and load balancing (β). 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 82%. The detail as to how these improvements are made is observed through the analysis of this case.

A cluster-level breakdown of these performance metrics is presented below. Three experimental cases are included, the results of which can be found in Table 3:

- Experiment 1: The case when the predictive management is inactive, the OFF case above;
- Experiment 2: Represents an intermediate case, the agent system is activated so global-level predictive management is performed, but local-level predictive scheduling (using the Titan system) remains inactive;
- Experiment 3: Predictive scheduling is used at the local level and prediction-based wide-area task management is used at the higher level, the ON case above.

The high request and submission rate imposes a large workload on each of the contributing clusters. The more powerful resources, C_0 and C_7, are better equipped to meet the deadlines for the tasks they receive than their less powerful counterparts, C_6 and C_{15}; this is reflected in ε. The resource utilisation rate (ν) of these powerful resources is also low (12 and 14%, as opposed to 54 and 51%).

Enabling the agent system (in experiment 2) increases the number of requests directed to these more powerful resources (ν of C_0 and C_7 increases to 55 and 46%), an effect which improves the overall system balance β to 81%. This improvement is also reflected in the new makespan (t) of 1551 seconds (a reduction of 37% over experiment 1).

Considerable further improvements can be made by employing predictive scheduling at the cluster level (in experiment 3). The resource utilisation rate ν of all resources improves to 77%, a marginal improvement is made to the system balance, yet there is a significant reduction in t to 438 seconds (an improvement of 82% over experiment 1).

These results are interesting in a number of ways. The use of predictive data for wide-area 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 cluster-level management is needed.

6. Conclusions

Performance services are set to play an important role in the management of resources in emerging wide-area, heterogeneous distributed computing environments. In order to support these services, tools are required to generate and publish performance data in a unified, yet decentralised manner. Performance data is likely to come in a number of forms, but by distributing this data, developers will be able to provide supporting high-level performance-based middleware services.

This paper documents how such services might be built. Raw 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 local level (demonstrated through the prediction-based Titan scheduler) and also at the level of wide-area task management (demonstrated through the supporting agent system).

A multi-tiered approach to this service provision is likely to prove successful. While an improvement in global load balancing will be achieved through the wide-area publishing and use of this performance data, it is only by using this information at local and global levels in the system that considerable performance gains will be achieved.

Acknowledgments

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

References

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

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

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

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

[5] R. Buyya, D. Abramson, and J. Giddy. Nimrod–G: An architecture for a resource management and scheduling system in a global computational grid. *Proc. of 4th Int. Conf. on High Performance Computing*, 2000.

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

[7] J. Cao, S. Jarvis, S. Saini, D. Kerbyson, and G. Nudd. ARMS: an agent-based resource management system for grid computing. *Scientific Programming, Special Issue on Grid Computing*, 10(2):135–148, 2002.

[8] J. Cao, D. Kerbyson, and G. Nudd. High performance service discovery in large-scale multi-agent and mobile-agent systems. *Int. J. of Software Engineering and Knowledge Engineering, Special Issue on Multi-Agent Systems and Mobile Agents*, 11(5):621–641, 2001.

[9] J. Cao, D. Kerbyson, E. Papaefstathiou, and G. Nudd. Performance modelling of parallel and distributed computing using PACE. *in Proceedings of 19th IEEE International Performance, Computing and Communication Conference (IPCCC'00)*, pages 485–492, 2000.

	Experiment 1			Experiment 2			Experiment 3		
C_i	ε(s)	ν(%)	β(%)	ε(s)	ν(%)	β(%)	ε(s)	ν(%)	β(%)
C_0	***-115***	***12***	***87***	***-611***	***55***	***91***	***-32***	***68***	***85***
C_1	-481	27	84	-567	58	87	-28	80	93
C_2	-710	37	83	-575	57	87	-53	92	94
C_3	-320	22	89	-309	40	88	32	65	95
C_4	-766	40	87	-538	55	87	-48	88	91
C_5	-1171	54	87	-577	54	83	-22	88	94
C_6	***-1161***	***54***	***90***	***-544***	***51***	***84***	***3***	***77***	***80***
C_7	***-137***	***14***	***85***	***-377***	***46***	***88***	***19***	***66***	***94***
C_8	-533	33	79	-437	49	79	53	74	90
C_9	-381	27	87	-540	54	90	-6	79	95
C_{10}	-381	24	82	-322	36	87	56	58	89
C_{11}	-996	53	83	-544	56	86	-13	87	92
C_{12}	-624	35	87	-598	53	89	-44	86	92
C_{13}	-946	49	87	-409	45	81	41	73	91
C_{14}	-976	46	85	-418	43	85	29	74	90
C_{15}	***-1097***	***51***	***83***	***-580***	***52***	***88***	***-2***	***75***	***86***
System	-681	36	58	-502	50	81	-6	77	84

Table 3. Cluster breakdown of experimental results for 1000 requests at 5 per second: In experiment 1 the predictive middleware (wide- and local-area) is off; in experiment 2 wide-area task management is performed but performance-driven local scheduling is not; in experiment 3 the predictive middleware is on. The results for the more powerful resources (C_0 and C_7) and less powerful resources (C_6 and C_{15}) are highlighted.

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

[11] S. Fitzgerald, I. Foster, and C. Kesselman. Grid information services for distributed resource sharing. *10th IEEE Int. Sym. on High-Performance Distributed Computing*, 2001.

[12] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *Intl J. of Supercomputer Applications*, 11(2):115–128, 1997.

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

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

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

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

[17] D. Kerbyson, H. Alme, A. Hoisie, F. Petrini, H. Wasserman, and M. Gittings. Predictive preformance and scalability modelling of a large-scale application. *Proceedings of Supercomputing '01*, 2001.

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

[19] M. Litzkow, M. Livny, and M. Mutka. Condor – a hunter of idle workstations. *Proceedings of 8th Int. Conf. on Distributed Computing Systems (IPDCS98)*, pages 104–111, 1998.

[20] W. McColl. Foundations of time-critical computing. *15th IFIP World Computer Congress, Vienna and Budapest*, 1998.

[21] G. Nudd, D. Kerbyson, E. Papaefstathiou, S. Perry, J. Harper, and D. Wilcox. PACE : A toolset for the performance prediction of parallel and distributed systems. *Int. J. of High Performance Computing Applications*, 14(3):228–251, 2000.

[22] S. Perry, R. Grimwood, D. Kerbyson, E. Papaefstathiou, and G. Nudd. Performance optimisation of financial option calculations. *Parallel Computing*, 26(5):623–639, 2000.

[23] D. Spooner, S. Jarvis, J. Cao, S. Saini, and G. Nudd. Local grid scheduling techniques using performance prediction. *IEE Proc., Comput. Digit. Tech.*, 2003.

[24] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima. Overview of a performance evaluation system for global computing scheduling algorithms. *Proc. of 8th IEEE Int. Symp. on High Performance Distributed Computing*, pages 97–104, 1999.

[25] J. Turner, D. Spooner, J. Cao, S. Jarvis, D. Dillenberger, and G. Nudd. A transaction definition language for Java application response measurement. *Journal of Computer Resource Measurement*, 105:55–65, 2001.

[26] R. Wolski, N. Spring, and J. Haye. The network weather service: A distributed resource performance forecasting service for metacomputing. *Future Generation Computing Systems*, 5-6:757–768, 1999.