ARMS: An agent-based resource management system for grid computing

Junwei Cao[a], Stephen A. Jarvis[a], Subhash Saini[b], Darren J. Kerbyson[c] and Graham R. Nudd[a]
[a]*Department of Computer Science, University of Warwick, Coventry, CV4 7AL, UK*
Tel.: +44 24 7652 2863; Fax: +44 24 7657 3024; E-mail: junwei@dcs.warwick.ac.uk
[b]*NASA Ames Research Centre, Moffett Field, California, USA*
[c]*Modelling, Algorithms, and Informatics Group, Los Alamos National Laboratory, USA*

Resource management is an important component of a grid computing infrastructure. The scalability and adaptability of such systems are two key challenges that must be addressed. In this work an agent-based resource management system, ARMS, is implemented for grid computing. ARMS utilises the performance prediction techniques of the PACE toolkit to provide quantitative data regarding the performance of complex applications running on a local grid resource. At the meta-level, a hierarchy of homogeneous agents are used to provide a scalable and adaptable abstraction of the system architecture. Each agent is able to cooperate with other agents and thereby provide service advertisement and discovery for the scheduling of applications that need to utilise grid resources. A case study with corresponding experimental results is included to demonstrate the efficiency of the resource management and scheduling system.

1. Introduction

Grid technologies have emerged to enable large-scale flexible resource sharing among dynamic virtual organisations [13,14]. An essential component of grid infrastructure software is the service layer, which acts as middleware between grid resources and grid applications. This work considers the resource management service, the component that provides efficient scheduling of applications utilising available resources in the grid environment [18]. Delivering such a service within the high performance community will rely, in part, on accurate performance prediction capabilities.

Previous research on the PACE (Performance Analysis and Characterise Environment) toolkit [20] can be used to provide quantitative data concerning the performance of sophisticated applications running on local high performance resources. PACE can supply accurate performance information for both the detailed analysis of an application and also as input to resource scheduling systems; this performance data can also be generated in real-time. While extremely well-suited for managing a locally distributed multi-computer, PACE functions do not map well onto wide-area environments, where heterogeneity, multiple administrative domains and communication irregularities increase the complexity of the resource management process. There are two key challenges that must be addressed:

– *Scalability.* A grid has the potential to encompass a large number of high performance computing resources. Each constituent of this grid will have its own function, its own resources and environment. These components are not necessarily fashioned to work together in the overall grid. They may be physically located in different organisations and may not be aware of each others capabilities.
– *Adaptability.* A grid is a dynamic environment where the location, type and performance of the components are constantly changing. For example, a component resource may be added to, or removed from, the grid at any time. These resources may not be entirely dedicated to the grid and therefore the computational capabilities of the system will vary over time.

An agent-based resource management system for grid computing, ARMS, is introduced to address the above challenges. Software agents are recognised as a powerful high-level abstraction for the modelling of complex software systems [16]. An agent-based methodology described in [5,8] is used to build large-scale distributed software systems that exhibit highly dynamic behaviour. It is intended that an entire system

Scientific Programming 10 (2002) 135–148
ISSN 1058-9244 / $8.00 © 2002, IOS Press. All rights reserved

be built of a hierarchy of identical agents with the same functionality. As such, agents are considered both service providers and service requestors and the implementation of system functions is abstracted to the processes of service advertisement and service discovery.

ARMS couples the performance prediction techniques of the PACE toolkit with a scheduling algorithm designed to manage a local grid resource. At the meta-level, ARMS utilises the agent-based methodology described in [7], where each agent acts as a representative for a local grid resource and considers this resource to be its high performance computing capability. Agents cooperate to perform service advertisement and discovery, thus providing the base services with which to manage and schedule applications over available grid resources. The performance of these agents can be improved by using a number of different optimisation strategies.

There are several solutions that currently address issues of resource management and scheduling. These include Globus [11], Legion [12], NetSolve [10], Condor [21], Ninf [19] and Nimrod/G [2]. While many of these projects utilise query-based mechanisms for resource discovery and advertisement [18], this work adopts an agent-based approach. This allows an agent to control the query process and to make resource discovery decisions based on its own internal logic as opposed to relying on a fixed-function query engine. Unlike Nimrod/G, in which the grid resource estimation is performed through heuristics and historical information, the performance prediction capabilities of grid resources in this research are achieved through the integration of PACE.

A number of recent grid projects have utilised existing distributed computing technologies such as CORBA [24] and Jini [1]. For example, the work described in [23] makes use of CORBA Lightweight Components to provide a new network-centred reflective component model which allows distributed applications to be assembled from independent binary components distributed on the network. The work described in [15] is a computational community that supports the federation of resources from different organisations; this system is designed and implemented in Java and Jini. While CORBA and Jini are well suited to their original design goals, they are not designed for developing high performance computing applications, and as mentioned in [14], such technologies only enable resource sharing within a single organisation.

An agent-based grid computing project is described in [22]. This work on an "Agent Grid", integrates services and resources for establishing multi-disciplinary problem solving environments. Specialised agents contain behavioural rules which can be modified based on their interaction with other agents and the environment in which they operate. In contrast, ARMS uses a hierarchy of homogenous agents for both service advertisement and discovery, and integrates these with a performance prediction based scheduler. A detailed introduction to this research can be found in [9].

The paper is organised as follows: the PACE toolkit is summarised in Section 2; the ARMS implementation is presented in Section 3; Section 4 describe a case study with corresponding experimental results and the paper concludes in Section 5.

2. The PACE toolkit

The main components of the PACE toolkit [4] are shown in Fig. 1. A core component of PACE is a performance specification language (PSL) which describes the performance aspects of an application and its parallelisation. A corresponding Hardware Modelling and Configuration Language (HMCL) is used to capture the definition of a computing environment in terms of its constituent performance model components and configuration information. The workload information and the component resource models are combined using an evaluation engine to produce time estimates and trace information of the expected application behaviour.

The performance prediction capabilities of PACE are demonstrated using the ASCI kernel application Sweep3D [3]. Table 1 shows the validation of the PACE model of Sweep3D against the code running on an SGI Origin2000 shared memory system. The accuracy of the prediction results are evaluated as follows:

$$\text{Error} = \frac{\text{Prediction} - \text{Measurement}}{\text{Measurement}} \times 100\%.$$

The maximum prediction error for this application is 11.44%, the average error is approximately 5%.

The key features of the PACE toolkit include: good level of predictive accuracy (approximately 15% maximum error), rapid evaluation time (typically seconds of CPU time) and a method for cross-platform comparison. These capabilities provide the basis for the application of PACE to dynamic grid environments consisting of a number of heterogeneous systems [17].

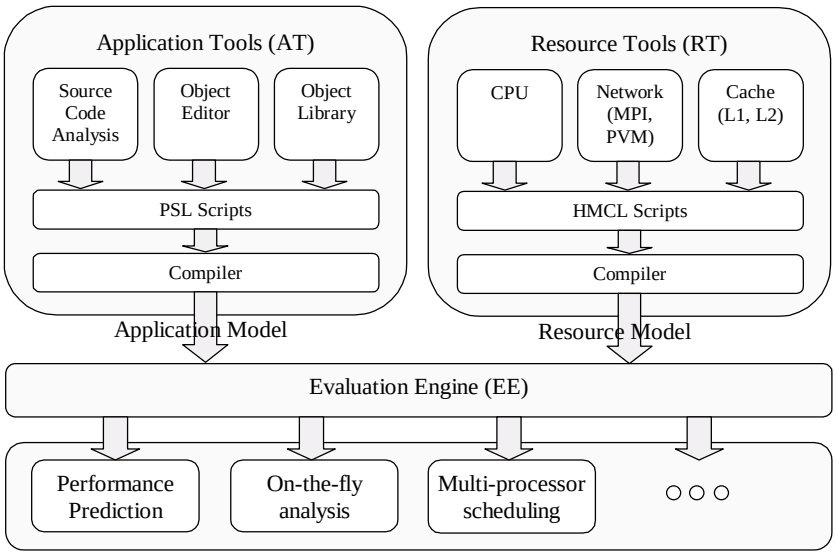

Fig. 1. The main components of the PACE toolkit include application tools, resource tools, and an evaluation engine. The rapid execution of PACE models lends itself to on-the-fly dynamic steering as well as traditional off-line performance prediction.

3. ARMS implementation

ARMS couples the agent-based methodology with the PACE performance prediction techniques in the implementation of grid resource management. The detail involved in this process is described below.

3.1. ARMS architecture

An overview of the ARMS architecture is illustrated in Fig. 2. The main components of this architecture include grid users, grid resources, ARMS agents and a performance monitor and advisor (PMA).

3.1.1. Grid users

There are a number of different categories of user of a grid computing environment. The grid users in Fig. 2, and who represent the main focus of this work, are considered to be scientists, who develop scientific high performance applications and use them to solve large problems in grid computing environments.

The user-side software primarily includes the PACE Application Tools. When a parallel application is developed, a corresponding application model is also produced. PACE performance modelling is an automated process, targeted at the non-professional performance engineer. When an application is submitted for execution, an associated performance model should also be attached.

Another component included in a grid request is the cost model, describing the user requirements concerning the application execution. This would include, for example, the deadline for the application to complete. Although there are a number of other metrics appropriate in this context, the current focus of this work is on execution time.

3.1.2. Grid resources

A grid resource provides high performance computing capabilities for grid users and might include supercomputers, or clusters of workstations or PCs.

In this system, PACE is used to create a hardware characterisation template that provides a model of each hardware resource. This characterisation is derived from computational and communication benchmarks which can be rapidly evaluated to provide dynamic performance data. The PACE hardware model is integral to the service information which is advertised across the agent hierarchy.

3.1.3. ARMS agents

Agents comprise the main components in the system; the agents are organised into a hierarchy and are designed to be homogenous. Each agent is viewed as a representative of a grid resource at a meta-level of resource management. This means that an agent can therefore be considered a service provider of high performance computing capabilities. The service information of each grid resource can be advertised within the agent hierarchy (in any direction) and agents can cooperate with each other to discover available resources.

Table 1
PACE model validation on an SGI Origin 2000. If the application has a data set of $15 \times 15 \times 15$ and is allocated to 9 processors (organised into a 3×3 processor array), each processor holds a total of $5 \times 5 \times 15$ data elements. Note that the results for single processor input are not included because there are many special configurations which are not included in the current performance model for the sequential code

Data size	2D Proc. Array	Prediction (s)	Measurement (s)	Err (%)
$15 \times 15 \times 15$	1×2	4.73037	4.440255	6.53
	2×2	2.59659	2.584936	0.45
	2×3	1.8373	1.812252	1.38
	2×4	1.51869	1.609818	−5.66
	3×3	1.3399	1.343736	−0.29
	3×4	1.10918	1.164072	−4.72
	4×4	0.907100	1.002728	−9.54
$25 \times 25 \times 25$	1×2	22.9501	20.780170	10.44
	2×2	12.1537	11.619632	4.60
	2×3	7.83574	7.893481	−0.73
	2×4	6.02865	5.979522	0.82
	3×3	5.52498	5.532116	−0.13
	3×4	4.24959	4.469564	−4.92
	4×4	3.36453	3.537966	−4.90
$35 \times 35 \times 35$	1×2	69.3858	64.832165	7.02
	2×2	36.1978	33.097098	9.37
	2×3	22.1074	21.160975	4.47
	2×4	16.3181	16.137180	1.12
	3×3	15.3466	15.272606	0.48
	3×4	11.3211	11.451001	−1.13
	4×4	8.84226	9.984213	−11.44
$50 \times 50 \times 50$	1×2	217.398	228.893311	−5.02
	2×2	112.307	102.285787	9.80
	2×3	65.6201	67.278086	−2.46
	2×4	46.7591	49.534483	−5.60
	3×3	45.1373	47.289627	−4.55
	3×4	32.1438	34.796392	−7.62
	4×4	24.8468	24.800020	0.20

Each agent utilises Agent Capability Tables (ACTs) to record service information of other agents. An ACT item is a tuple containing an agent ID and corresponding service information – all performance related information of a grid resource which can be used in the estimation of its performance.

An agent can choose to maintain different ACTs corresponding to the different sources of service information: T_ACT is used to record service information of local resources; L_ACT is used to record service information received from lower agents in the hierarchy; G_ACT to record information from the upper agent in the hierarchy; finally, C_ACT is used to store cached service information.

There are two methods of maintaining ACT coherency – data-pull and data-push, each of which occur periodically or can be driven by system events:

– *Data-pull* – An agent asks other agents for their service information either periodically or when a request arrives.
– *Data-push* – An agent submits its service information to other agents in the system periodically or when the service information is changed.

An agent uses the ACTs as a knowledge base. This is used to assist in the service discovery process triggered by the arrival of a request. Service discovery involves querying the contents of the ACTs in the order: T_ACT, C_ACT, L_ACT and G_ACT. If an agent exhausts the ACTs, and does not obtain the required service information, it can submit the request to its upper agent or terminate the discovery process.

The PACE evaluation engine is integrated into each agent. Its performance prediction capabilities are used for local resource management in the scheduling of parallel applications over available local processors. The evaluation engine is also used to provide support to the service discovery process.

The agent system aims to bridge the gap between grid users and resources and in so doing, allows the efficient scheduling of applications over available grid resources. An agent can select different strategies of ser-

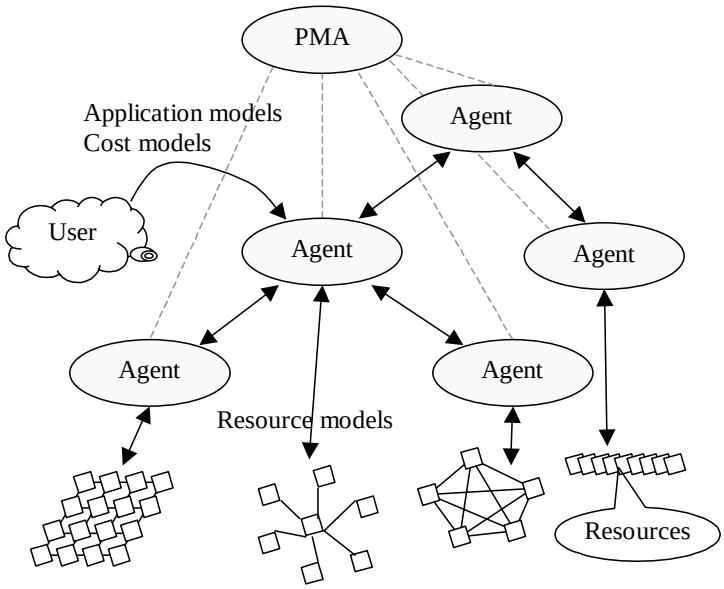

Fig. 2. ARMS architecture. The main components include users, resources, agents and an agent performance monitor and advisor (PMA).

vice advertisement and discovery, the choice of which may lead to different performance outcomes.

3.1.4. ARMS PMA

A special agent, illustrated in Fig. 2, is capable of modelling and simulating the performance of the agent system while the system is active. This is known as the performance monitor and advisor (PMA) of the system.

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 nor serves as a communication centre in the physical and symbolic sense. If the PMA ceases to function, the agent system has no operational difficulties and continues with ordinary system behaviour. Efficiency improvements to the agent system are only made possible through the modelling and simulation mechanism built into the PMA. The PMA also avoids any one agent in the system becoming a single system bottleneck.

Statistical data is monitored from each of the agents and input to the PMA for performance modelling. The performance model is processed by the simulation engine in the PMA so that new optimisation strategies can be chosen and the performance metrics improved. The process of simulation allows a number of strategies to be explored until a better solution is selected. The selected optimisation strategies are then returned and used to reconfigure the agents in the system. A detailed account of the structure and function of the PMA can be found in [6].

3.2. ARMS agent structure

The agent structure in ARMS is shown in Fig. 3. Each layer has several modules, which cooperate with each other to perform service advertisement, service discovery, and application execution. The three layers are discussed below.

The communication layer of each agent performs communication functions and acts as an interface to the external environment. From the communication module, an agent can receive both service advertisement and discovery messages. It interprets the contents of each message and submits the information to corresponding modules in the coordination layer of the agent. For example, an advertisement message from another agent will be directly sent to the ACT manager in the agent coordination layer. The communication module is also responsible for sending service advertisement and discovery messages to other agents.

There are four components in the coordination layer of an agent: the ACT manager, the PACE evaluation engine, a scheduler and a matchmaker. These work together to make decisions as to how an agent should act on the receipt of messages from the communication layer. For example, the final response to a service discovery message would involve application execution on the local resource or the dispatching of the request to another agent.

The main functions of local resource management in an agent include application management, resource al-

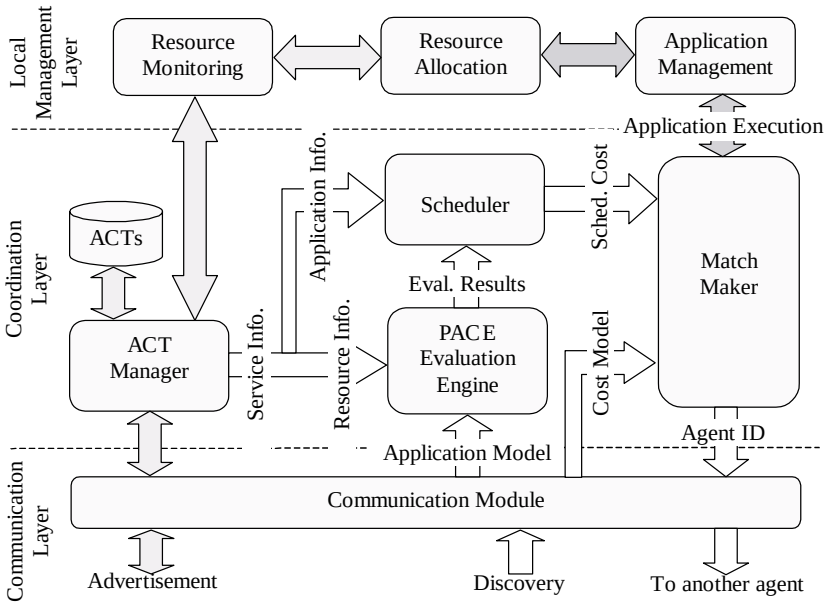

Fig. 3. ARMS agent structure. The greyscale of the arrows indicate three main processes: service advertisement, service discovery and application execution.

location and resource monitoring. Application execution commands are sent from the coordination layer to the local agent manager, these commands include the scheduling information for an application (start time, allocated processor ids etc.). The Application Management part of the system is also responsible for managing the queuing of applications that have been scheduled to be executed on the locally managed resources. At the start time an application is dispatched to the Resource Allocation component. Resource allocation includes wrappers for different application execution environments including MPI and PVM; it is at this stage that the application is actually executed on the local scheduled processors. Another important component of local resource management is resource monitoring. This is responsible for controlling the PACE benchmark programs which are executed on the local resource and from which corresponding resource models are dynamically created. The resource monitor is also responsible for communicating other resource and application information between the application management and resource allocation modules. It also coordinates all the collected information concerning local resources into service information which is then reported to the T_ACT in the coordination layer of the agent.

These agent functions are described in detail below. In particular, the implementation of the agent coordination layer is emphasised and the four main components of the scheduling algorithm are documented.

3.2.1. ACT manager

The ACT manager controls agent access to the ACT database, where service information regarding grid resources is located. Figure 4 illustrates the content of this service information.

Consider a grid resource with n processors where each processor P_i has its own type ty_i. A PACE hardware model can be used to describe the performance information of a processor. The processors of a grid resource can be expressed as follows:

$$P = \{P_i | i = 1, 2, \ldots, n\}$$
$$ty = \{ty_i | i = 1, 2, \ldots, n\}.$$

Let m be the number of applications that are running, or being queued to be executed on a grid resource. Each application A_j has two attributes – scheduled start time ts_j and end time te_j. The applications of a grid resource can then be expressed as follows:

$$A = \{A_j | j = 1, 2, \ldots, m\}$$
$$ts = \{ts_j | j = 1, 2, \ldots, m\}$$
$$te = \{te_j | j = 1, 2, \ldots, m\}.$$

Let MA_j be the set of processors that are allocated to application A_j:

$$MA = \{MA_j | j = 1, 2, \ldots, m\}$$
$$MA_j = \{P_{i_l} | l = 1, 2, \ldots, k_j\},$$

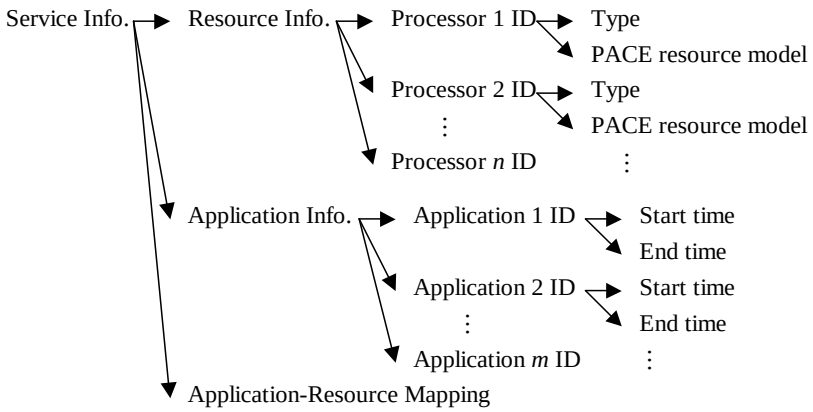

Fig. 4. Service information in ARMS. Each processor is described using the corresponding PACE resource model. The scheduled start and end time for the application executions are also included. The mapping between applications and resources is represented using a 2D array.

where k_j is the number of processors that are allocated to application A_j. Let M be a 2D array, which describes the mapping relationships between resources and applications using boolean values.

$$M = \{M_{ij} | i = 1, 2, \ldots, n;\ j = 1, 2, \ldots, m\}$$

$$M_{ij} = \begin{cases} 1 & \text{if } P_i \in MA_j \\ 0 & \text{if } P_i \notin MA_j \end{cases}$$

3.2.2. PACE evaluation engine

In ARMS, a request for service discovery involves finding an available grid resource for an application. The request information is composed of the PACE application model am, which includes all of the performance related information of an application A_r. The application model is one of the inputs to the PACE evaluation engine found in an agent.

The requirements of an application is specified using a cost model. This model includes metrics such as the deadline for the execution of an application, t_{req}, and is used as one of the inputs to the matchmaker part of the agent system.

The PACE evaluation engine has two inputs, firstly the application model (am) from the service discovery request, and secondly the resource information (ty) from the ACT manager. Using this information, the PACE evaluation engine can produce performance prediction data including the expected execution time ($exet$) necessary for the application to be executed on the given resource.

$$exet = eval(ty, am)$$

Rather than running the application on all the available processors of a grid resource P, an application can be executed on any subset of processors $\overline{P}$ (note that $\overline{P}$ cannot be the empty set Φ). This is expressed as follows:

$$\forall \overline{P} \subseteq P,\ \overline{P} \neq \Phi,\ \overline{ty} \subseteq ty,\ \overline{ty} \neq \Phi,$$
$$\overline{evet} = eval(\overline{ty}, am).$$

The output of the PACE evaluation engine ($exet$) forms one of the inputs to the scheduler of the agent. Another input to the scheduler is the application information from an ACT item.

3.2.3. Scheduler

An ACT item acts as a view of a grid resource that is remote to the agent. An agent can however still schedule the required application execution based on this information of a resource. The function of the scheduler is to find the earliest time at which the application will terminate, a function described by the ACT item t_{sched}.

$$t_{sched} = \min_{\forall \overline{P} \subseteq P, \overline{P} \neq \Phi} \left(\overline{te_r}\right)$$

The application has the possibility of being allocated to any selection of processors comprising a grid resource. The scheduler considers all these possibilities and chooses the earliest end time for the execution. This end time – equal to the earliest possible start time plus the total execution time – is described as follows:

$$\overline{te_r} = \overline{ts_r} + \overline{exet}.$$

The earliest possible start time for application A_r to be executed on a selection of processors $\overline{P}$, is defined as the time at which all of these processors become free. If all of these processors are already idle, then the application can be executed immediately. This figure can be expressed as follows:

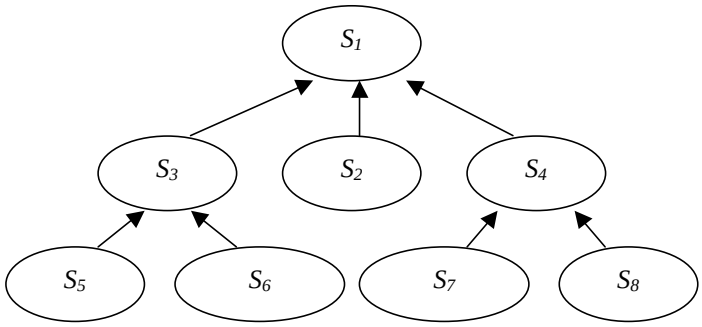

Fig. 5. Case study: agent hierarchy. The agent at the head of the hierarchy is S_1, which has three lower agents, S_2, S_3 and S_4. Agent S_2 has no lower agents, while S_3 and S_4 have two lower agents each.

$$\overline{ts_r} = \max\left(t, \max_{\forall i, P_i \in \overline{P}} (td_i)\right),$$

where td_i is the latest free time of processor P_i. This is equivalent to the maximum end time of the applications that are allocated to process P_i:

$$td_i = \max_{\forall j, M_{ij}=1} (te_j).$$

In summary, t_{sched} can be calculated as follows:

$$T_{sched} = \min_{\forall \overline{P} \subseteq P, \overline{P} \neq \Phi} \left(\max\left(t, \max_{\forall i, P_i \in \overline{P}} \left(\max_{\forall j, M_{ij}=1} (te_j)\right)\right) + \overline{exet}\right).$$

It is not necessarily the case that scheduling all processors to an application will achieve higher performance. For example, the start time of application execution may be earlier if only a number of processors are selected; similarly, the execution of some applications may take longer if too many processors are allocated to the task.

The scheduling algorithm described above is used in the implementation of ARMS. The complexity of the algorithm is determined by the number of possible processor selections:

$$C_n^1 + C_n^2 + \ldots + C_n^n = 2^n - 1.$$

It is therefore essential that the evaluation engine supplied with PACE is efficient. During each scheduling process, the evaluation function can be called $2^n - 1$ times. Even in the situation where all the processors of a grid resource are of the same type, the evaluation function still needs to be called n times. PACE evaluation can be performed very quickly to produce prediction results on the fly; this is a key feature of PACE which enables the toolkit to provide service discovery support for ARMS.

Table 2
Case study: resources. Each resource is composed of 16 processors (for SGI) or hosts (for Sun), and each host is of the same type

Agent	Resource type	#Processors/Hosts
S_1	SGI Origin 2000	16
S_2	SGI Origin 2000	16
S_3	Sun Ultra 10	16
S_4	Sun Ultra 10	16
S_5	Sun Ultra 1	16
S_6	Sun Ultra 5	16
S_7	Sun SPARCstation 2	16
S_8	Sun SPARCstation 2	16

3.2.4. Matchmaker

The matchmaker in an agent is responsible for comparing the scheduling results with the cost model attached to the request. The comparison results lead to different decisions on agent behaviours.

In terms of application execution time, if $t_{req} \geqslant t_{sched}$, then the corresponding resource can meet the users requirement. If the corresponding ACT item is in the T_ACT, a local resource is available (and capable) of executing the application. In this case the application execution command is sent to the local manager in the agent, or the agent ID of the corresponding ACT item is returned and the agent dispatches the request to the agent via the agent ID.

If $t_{req} < t_{sched}$, the corresponding resource cannot meet the requirement of the user. The agent continues to look up other items in the ACTs until the available service information is found. If there is no further service information available in the ACTs, the agent may submit or dispatch the request to upper or lower agents. This instantiates further service discovery governed by the service discovery strategy implemented by the agent.

ARMS demonstrates how an agent-based methodology coupled with the prediction capabilities of PACE, provides a system of resource management for grid computing. A case study of this system is given below.

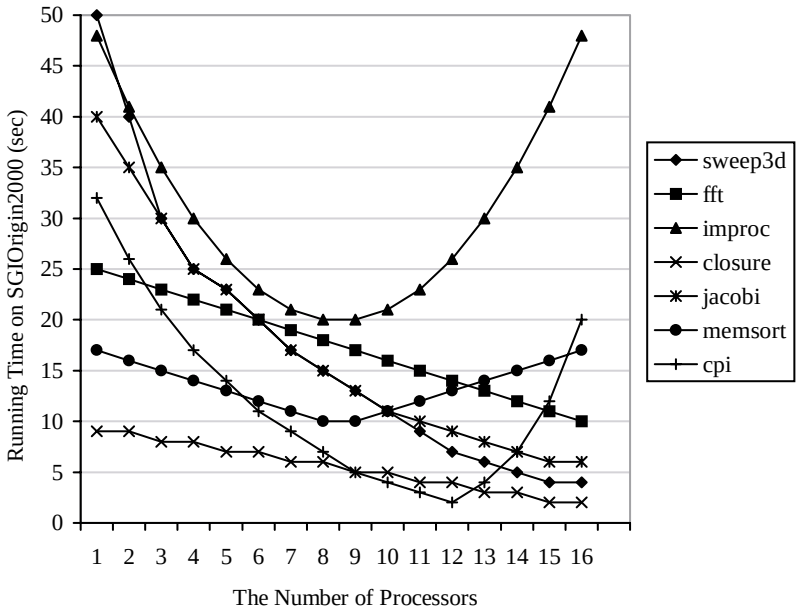

Fig. 6. Case study: Applications. The timings shown in the figure are PACE performance prediction results for the execution of these applications on an SGI Origin2000.

4. A case study

In this section experimental results are documented which show how ARMS schedules applications onto available resources. There are two main parts in the design of the experiments. ARMS is configured to include agents, resources and agent behaviour strategies. The sending of application requests (via 'virtual users') is automated, this is so that execution requests can be sent to ARMS with varying frequencies, thus simulating different workloads on the system.

4.1. System design

There are 8 agents in the experimental system. The agent hierarchy is shown in Fig. 5. Each agent represents a local grid resource and information describing the capabilities of the resources is shown in Table 2. The SGI multi-processor is the most powerful resource, followed by the Sun Ultra 10, 5, 1 and the SparcStation.

In the experimental system, the T_ACT, L_ACT and G_ACT are used in each agent. T_ACTs are maintained by event-driven data-push service advertisement. L_ACTs are updated once every 10 seconds using a periodical data-pull. G_ACTs are updated once every 30 seconds using a periodical data-pull. All agents use the same strategy except S_1, which found at the head of the agent hierarchy, does not maintain a G_ACT.

Table 3
Case study: requirements. For example, a required execution time for the application sweep3d will be chosen at random between 4 s and 200 s, when a request is sent to ARMS

Application	Minimum requirement (s)	Maximum requirement (s)
sweep3d	4	200
fft	10	100
improc	20	192
closure	2	36
jacobi	6	160
memsort	10	68
cpi	2	128

Table 4
Case study: workloads. For example, experiment No. 2 lasts approximately 7 minutes. During this period, a total of 149 requests are sent to ARMS; one request is sent every 3 seconds on average

Experiment no.	1	2	3	4
Minimum request interval (s)	1	1	1	1
Maximum request interval (s)	7	5	3	1
Average frequency (s/app)	4	3	2	1
Experiment last time (min)	7	7	7	5
Total application number	109	149	215	293

4.2. Virtual users

The applications used in the experiments are typical scientific computing programs; these include sweep3d, fft, improc, closure, jacobi, memsort and cpi. Each application is modelled and evaluated using the PACE toolkit. The performance evaluation results against the

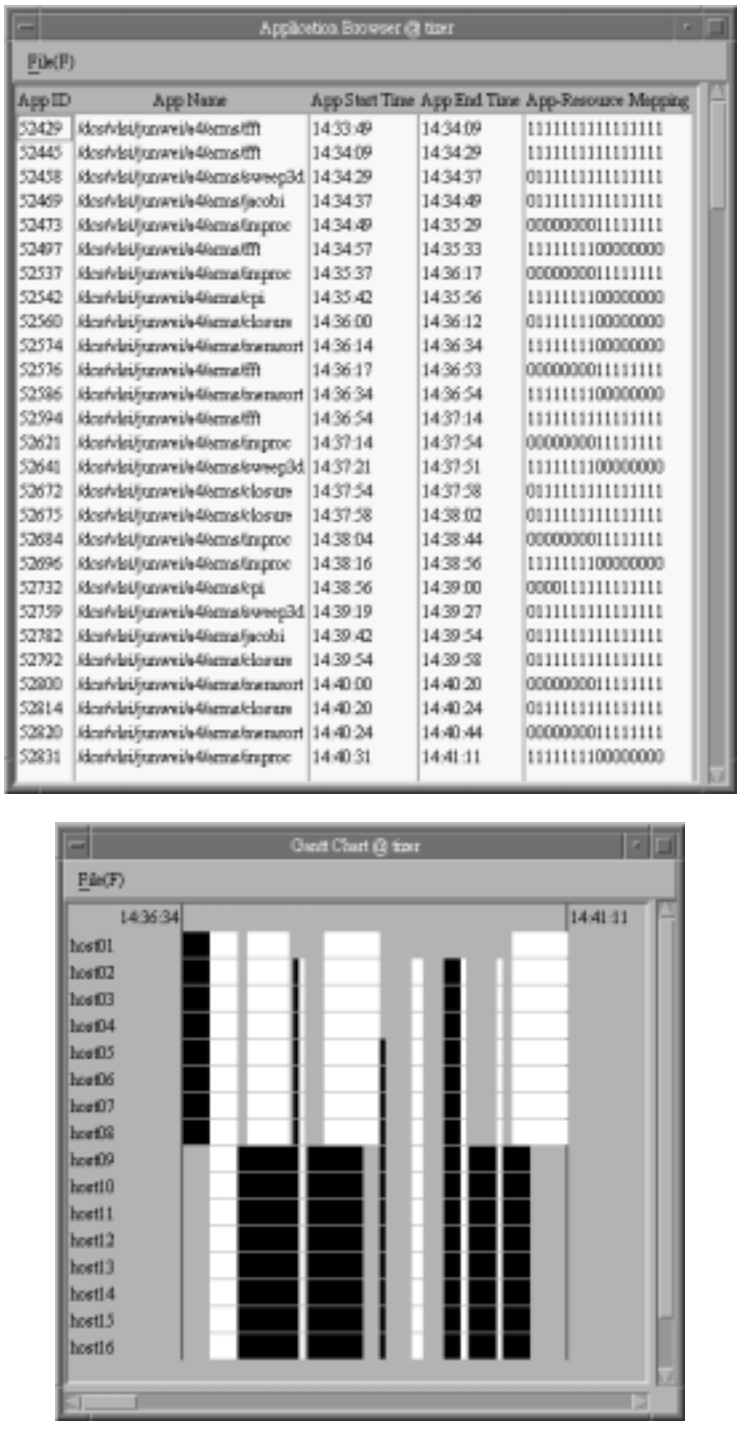

Fig. 7. Experimental results: No. 2 @ S_4. In the experiment No. 2, 27 of the 149 requests are scheduled to be executed using the resource of S_4. The latest 16 applications are also illustrated using the Gantt chart according to the scheduling information in the list above.

SGI Origin2000 can be found in Fig. 6. The run time of the applications on other platforms is greater than that of the SGI Origin2000, the general trend however is similar and therefore these figures are not documented.

An application execution request for one of the seven test applications is sent at random to an agent. Addi-

Table 5
Experimental results: the distribution of applications on agents. In experiment No. 2, 27 application execution requests are scheduled onto the resources of agent S_4 (this conforms to the results shown in Fig. 7) corresponding to 19 percent of the total 149 requests. 5 requests (3 percent of the total) are not scheduled onto any resource and end unsuccessfully

Agent	Experiment number							
	1		2		3		4	
	No.	%	No.	%	No.	%	No.	%
S_1	13	12	27	19	45	21	45	15
S_2	13	12	15	10	27	13	42	14
S_3	15	14	20	13	27	13	38	13
S_4	14	13	27	19	31	14	39	13
S_5	10	9	15	10	20	9	28	10
S_6	13	12	17	11	23	11	31	11
S_7	14	13	12	8	16	7	26	9
S_8	14	13	11	7	17	8	24	8
Failed	3	2	5	3	9	4	20	7
Total	109	100	149	100	215	100	293	100

Table 6
Experimental results: service discovery. In experiment No. 2, the resources for 114 application execution requests are satisfied by the agent they are submitted to first; representing 77 percent of the total 149 requests. Three agents are involved in 2-step service discovery. The first agent receives the request from the user, a second acts as a go-between to a third agent at which the corresponding resource is found

Step	Experiment number							
	1		2		3		4	
	No.	%	No.	%	No.	%	No.	%
0-step	106	97	114	77	143	66	199	68
1-step	3	3	24	16	38	18	29	10
2-step	0	0	11	7	31	15	53	18
3-step	0	0	0	0	3	1	12	4
Total	109	100	149	100	215	100	293	100

tionally, the required execution time for the application is also selected randomly from a given domain, this can be found in Table 3.

The automatic users are configured to send requests at different frequencies. Table 4 documents four ARMS experiments whose design is based on the varying workloads of the system. The interval at which requests are sent is chosen randomly from a given domain, this results in a different average frequency of requests for each experiment. The experimental results are discussed in the following section.

4.3. Experimental results

Experiment No. 2 lasts approximately 7 minutes. During this period 149 requests are sent to ARMS and scheduled on the eight available resources. An example agent view is given in Fig. 7. The detailed results for the other agent views and experiments are not given but are summarised as statistical data included in Tables 5 and 6, and illustrated in Figs 8 and 9 respectively. The curves in the figures show the trend of application distribution when the system workload increases; this detail is discussed:

Experiment No. 1

In experiment No. 1, one request is sent every 4 seconds on average. Application execution requests are sent out to the agents randomly, ensuring that each agent should receive approximately the same number of requests from the users. In this experiment the system workload is light relative to the capabilities of the resources (even for the resources associated with agents S_7 and S_8). The results show that 97 percent of applications require 0-step resource discovery, i.e. the majority of the requests are met by the agents to which the requests first arrive. Almost no service discovery is required between agents. This results in an average distribution of applications to agents and the number of requests that end unsuccessfully is very small.

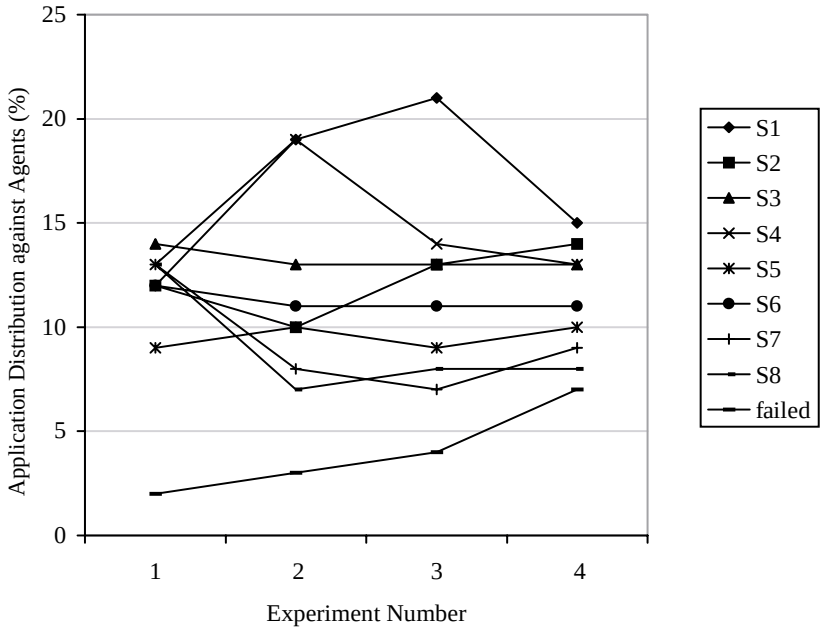

Fig. 8. Experimental results: application execution distribution on agents when system workloads increase. The percent of failed requests for each experiment are also shown; note that these increase with the system workload.

Experiment No. 2

When the system workload becomes heavier, many requests that S_7 and S_8 cannot meet are submitted to their upper agent S_4. This leads to a heavy workload on S_4 (19% of total application executions). The resources provided by agents S_5 and S_6 are more powerful. However, they still cannot meet all the requests from users. Some of the requests are submitted to their upper agent S_3, this also leads to a heavy workload on S_3, though less so than on S_4. Consequently there is a dramatic increase in the number of 1-step service discovery processes.

The system is configured so that the agent at the head of the agent hierarchy, S_1, represents the most powerful computing platform (a multi-processor SGI Origin2000). There are some application requests that have time-critical requirements and which are only met using the SGI Origin2000. These requests are also submitted from S_4 or S_3 to S_1. This leads to a heavy workload on S_1 and also increases the process of 2-step service discovery. The agent S_2 also represents a powerful resource (as that of S_1) and also meets the requirements of the requests it receives from the users. However, S_2 is topologically distant from the other agents and as a result S_2 remains under utilised.

Experiment No. 3

The system workload is increased further. The dramatic decrease in the percentage of application executions on S_4 indicates that this local resource has reached capacity. Many of the requests submitted from S_7 and S_8 have to be passed to S_1, this leads to a dramatic increase of the number of 2-step discovery processes. The number of 1-step discovery processes also increases and 3-step discovery processes begin to emerge. More application executions are scheduled onto the agent S_2. All of these indicate that service discovery among the agents becomes more active when the system workload increases.

Experiment No. 4

This experiment represents a heavy workload. The decrease in the percentage of application executions on S_1 indicates that the local resource S_1 also reaches capacity; this also signals an increase in the number of failed requests. The number of 1-step discovery processes decreases, while 2-step and 3-step service discovery processes increase. All of these indicate that the whole system has reached its capacity and so more complex service discovery processes are common. In this situation, the distribution of applications over the agents appears well balanced. The workload of the agents also mirrors the computing capabilities of their resources. The agents S_1 and S_2, which represent the most powerful resources in the experiment system, are assigned more applications, this is followed by S_3, S_4,

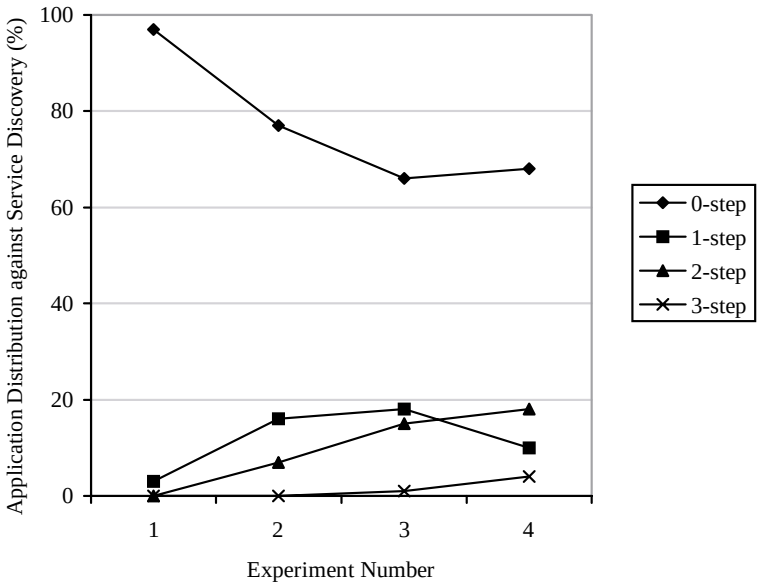

Fig. 9. Experimental results: service discovery. When the system workload increases, the number of 0-step service discovery decreases and those corresponding to more complex service discovery processes increase.

S_6 and S_5. Only a small number of requests are met at the agents S_7 and S_8.

The results show that the coupling of performance prediction and agent-based service advertisement and discovery is effective for the management and scheduling of grid resources.

Scalability and adaptability are two key challenges in grid resource management. The case study described in this paper is far from grid-sized. However, the experimental results demonstrate that the agents in ARMS only need communicate with their neighbouring agents. The process of service discovery is achieved through the transitive closure of these step-wise requests, a feature which makes it possible for the system to scale when the grid environment becomes large.

Another important factor which allows ARMS to achieve high performance, is the capability of agents to adjust their service advertisement and discovery strategies in order to adapt to the highly dynamic grid environment. The choice of different strategies impacts on the service discovery performance of the overall system; results of which are discussed in [6].

5. Conclusions

In this paper an agent-based grid resource management system, ARMS, is implemented using a hierarchy of homogenous agents coupled with a performance prediction toolkit. Experimental results are included, demonstrating the efficiency of ARMS in the scheduling of grid applications over available grid resources.

Future work is underway on a practical implementation of this grid resource management system. A transaction-based performance modelling technique [25,26] is under development which can be used to achieve remote performance prediction more efficiently. A prediction-driven distributed grid resource scheduler is also being developed based on an iterative heuristic algorithm. The supporting agent system is now Java based and agent cooperation is implemented via an XML agent communication language.

Acknowledgements

This work is sponsored in part by grants from the NASA AMES Research Centre (administered by USARDSG, contract no. N68171-01-C-9012) and the EPSRC (contract no. GR/R47424/01).

References

[1] K. Amold, B. O'Sullivan, R. Scheifer, J. Waldo and A. Woolrath, The Jini specification, Addison Wesley, 1999.

[2] R. Buyya, D. Abramson and J. Giddy, Nimrod/G: an architecture for a resource management and scheduling system in a global computational grid, in: *Proceedings 4th International Conference on High Performance Computing in Asia-Pacific Region*, Beijing, China, 2000.

[3] J. Cao, D.J. Kerbyson, E. Papaefstathiou and G.R. Nudd, Modelling of ASCI high performance applications using PACE, in: *Proceedings 15th Annual UK Performance Engineering Workshop*, Bristol, UK, 1999, pp. 413–424.

[4] J. Cao, D.J. Kerbyson, E. Papaefstathiou and G.R. Nudd, Performance modeling of parallel and distributed computing using PACE, in: *Proceedings 19th IEEE International Performance, Computing and Communication Conference*, Phoenix, USA, 2000, pp. 485–492.

[5] J. Cao, D.J. Kerbyson and G.R. Nudd, Dynamic application integration using agent-based operational administration, in: *Proceedings 5th International Conference on Practical Application of Intelligent Agents and Multi-Agent Technology*, Manchester, UK, 2000, pp. 393–396.

[6] J. Cao, D.J. Kerbyson and G.R. Nudd, Performance evaluation of an agent-based resource management infrastructure for grid computing, in: *Proceedings 1st IEEE/ACM International Symposium on Cluster Computing and the Grid*, Brisbane, Australia, 2001, pp. 311–318.

[7] J. Cao, D.J. Kerbyson and G.R. Nudd, Use of agent-based service discovery for resource management in metacomputing environment, in: *Proceedings of 7th International Euro-Par Conference, LNCS 2150*, Manchester, UK, 2001, pp. 882–886.

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

[9] J. Cao, *Agent-based resource management for grid computing*, Ph.D. Dissertation, University of Warwick, 2001.

[10] H. Casanova and J. Dongarra, Applying NetSolve's network-enabled server, *IEEE Computational Science & Engineering* **5**(3) (1998), 57–67.

[11] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith and S. Tuecke, A resource management architecture for metacomputing systems, in: *Proceedings IPPS/SPDP Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

[12] S.J. Chapin, D. Katramatos, J. Karpovich and A. Grimshaw, Resource management in Legion, *Future Generation Computer Systems* **15**(5) (1999), 583–594.

[13] I. Foster and C. Kesselman, *The GRID: blueprint for a new computing infrastructure*, Morgan-Kaufmann, 1998.

[14] I. Foster, C. Kesselman and S. Tuecke, The anatomy of the grid: enabling scalable virtual organizations, to appear in: *Int. J. Supercomputer Applications*, 2001.

[15] N. Furmento, S. Newhouse and J. Darlington, Building computational communities from federated resources, in: *Proceedings of 7th International Euro-Par Conference, LNCS 2150*, Manchester, UK, 2001, pp. 855–863.

[16] N.R. Jennings and M.J. Wooldridge, eds, *Agent technology: foundations, applications, and markets*, Springer-Verlag, 1998.

[17] D.J. Kerbyson, J.S. Harper, E. Papaefstathiou, D.V. Wilcox and G.R. Nudd, Use of performance technology for the management of distributed systems, in: *Proceedings 6th International Euro-Par Conference, LNCS 1900*, Germany, 2000, pp. 149–159.

[18] K. Krauter, R. Buyya and M. Maheswaran, A taxonomy and survey of grid resource management systems, to appear in: *Software: Practice and Experience*, 2001.

[19] H. Nakada, H. Takagi, S. Matsuoka, U. Nagashima, M. Sato and S. Sekiguchi, Utilizing the metaserver architecture in the Ninf global computing system, in: *Proceedings High-Performance Computing and Networking Europe, LNCS 1401*, Amsterdam, 1998, pp. 607–616.

[20] G.R. Nudd, D.J. Kerbyson, E. Papaefstathiou, S.C. Perry, J.S. Harper and D.V. Wilcox, PACE – a toolset for the performance prediction of parallel and distributed systems, *Intl. J. High Performance Computing Applications, Special Issues on Performance Modelling – Part I* **14**(3) (2000), 228–251.

[21] R. Raman, M. Livny and M. Solomon, Matchmaking: distributed resource management for high throughput computing, in: *Proceedings 7th IEEE International Symposium on High Performance Distributed Computing*, Chicago, Illinois, July 1998.

[22] O.F. Rana and D.W. Walker, The Agent Grid: agent-based resource integration in PSEs, in: *Proceedings 16th IMACS World Congress on Scientific Computation*, Applied Mathematics and Simulation, Lausanne, Switzerland, 2000.

[23] D. Sevilla, J.M. García and A. Gómez, CORBA lightweight components: a model for distributed component-based heterogeneous computation, in: *Proceedings of 7th International Euro-Par Conference, LNCS 2150*, Manchester, UK, 2001, pp. 845–854.

[24] D. Slama, J. Garbis and P. Russell, *Enterprise CORBA*, Prentice Hall, 1999.

[25] D.P. Spooner, J.D. Turner, J. Cao, S.A. Jarvis and G.R. Nudd, Application characterisation using a lightweight transaction model, in: *Proceedings 17th Annual UK Performance Engineering Workshop*, Leeds, UK, 2001, pp. 215–225.

[26] J.D. Turner, D.P. Spooner, J. Cao, S.A. Jarvis, D.N. Dillenberger and G.R. Nudd, A transaction definition language for Java application response measurement, *J. Computer Resource Management* **105** (2002), 55–65.