Available online at www.sciencedirect.com
SCIENCE DIRECT®
Future Generation Computer Systems 21 (2005) 135–149
FGCS FUTURE GENERATION COMPUTER SYSTEMS
www.elsevier.com/locate/future

Grid load balancing using intelligent agents

Junwei Cao[a,*,1], Daniel P. Spooner[b], Stephen A. Jarvis[b], Graham R. Nudd[b]

[a] *Center for Space Research, Massachusetts Institute of Technology, Cambridge, MA, USA*
[b] *Department of Computer Science, University of Warwick, Coventry, UK*

Available online 28 October 2004

Abstract

Scalable management and scheduling of dynamic grid resources requires new technologies to build the next generation intelligent grid environments. This work demonstrates that AI techniques can be utilised to achieve effective workload and resource management. A combination of intelligent agents and multi-agent approaches is applied to both local grid resource scheduling and global grid load balancing. Each agent is a representative of a local grid resource and utilises predictive application performance data with iterative heuristic algorithms to engineer local load balancing across multiple hosts. At a higher level, agents cooperate with each other to balance workload using a peer-to-peer service advertisement and discovery mechanism.

Keywords: Load balancing; Grid computing; Intelligent agents; Genetic algorithm; Service discovery

1. Introduction

Grid computing originated from a new computing infrastructure for scientific research and cooperation [36,20] and is becoming a mainstream technology for large-scale resource sharing and distributed system integration [21]. Current efforts towards making the global infrastructure a reality provide technologies on both grid services and application enabling [6].

Workload and resource management are essential functions provided at the service level of the grid software infrastructure. Two main challenges that must be addressed are scalability and adaptability. Grid resources are geographically distributed and resource performance can change quickly over time. Grid users submit tasks with different resource and quality of service (QoS) requirements. For management and scheduling to be effective, such systems must develop intelligent and autonomous decision-making techniques.

Software agents have been accepted to be a powerful high-level abstraction for modelling of complex software systems [26]. In our previous work, an agent-based methodology is developed for building large-scale distributed systems with highly dynamic behaviours [9,10]. This has been used in the implementation of an agent-based resource management system for metacomputing [11] and grid computing [12–14].

* Corresponding author.
E-mail address: caoj@mit.edu (J. Cao).
[1] This work was carried out when the author was with the University of Warwick.

0167-739X/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.future.2004.09.032

This work focuses on grid load-balancing issues using a combination of both intelligent agents and multi-agent approaches. Each agent is responsible for resource scheduling and load balancing across multiple hosts/processors in a local grid. The agent couples application performance data with iterative heuristic algorithms to dynamically minimise task makespan and host idle time, while meeting the deadline requirements for each task. The algorithm is based on an evolutionary process and is therefore able to absorb system changes such as the addition or deletion of tasks, or changes in the number of hosts/processors available in a local grid.

At the global grid level, each agent is a representative of a grid resource and acts as a service provider of high performance computing power. Agents are organised into a hierarchy and cooperate with each other to discover available grid resources for tasks using a peer-to-peer mechanism for service advertisement and discovery.

Agents are equipped with existing PACE application performance prediction capabilities [8,32]. The key features of the PACE toolkit include a good level of predictive accuracy, rapid evaluation time and a method for cross-platform comparison. These features enable PACE performance data to be utilized in real-time for agents to perform grid resource scheduling [15,16].

Several metrics are considered to measure the load-balancing performance of grid agents. A case study is included and corresponding results conclude that intelligent agents, supported by application performance prediction, iterative heuristic algorithms and service discovery capabilities, are effective to achieve overall resource scheduling and load balancing, improve application execution performance and maximise resource utilisation.

2. Grid agents

This work combines intelligent agents and multi-agent approaches. The agent structure and hierarchy are described below.

2.1. Agent structure

Each agent is implemented so it can manage hosts/processors for a local grid resource, scheduling incoming tasks to achieve local load balancing. Each agent provides a high-level representation of a grid resource and therefore characterises these resources as high performance computing service providers in a wider grid environment. The layered structure of each agent is explained below:

- *Communication layer*. Agents in the system must be able to communicate with each other or with users using common data models and communication protocols. The communication layer provides an agent with an interface to heterogeneous networks and operating systems.
- *Coordination layer*. The request an agent receives from the communication layer should be explained and submitted to the coordination layer, which decides how the agent should act on the request according to its own knowledge. For example, if an agent receives a service discovery request, it must decide whether it has related service information. This is described in detail in Section 4.
- *Local management layer*. This layer performs functions of an agent for local grid load balancing. Detailed scheduling algorithms are described in Section 3. This layer is also responsible for submitting local service information to the coordination layer for agent decision making.

2.2. Agent hierarchy

Agents are organised hierarchically in a higher level global grid environment, as shown in Fig. 1. The broker is an agent that heads the whole hierarchy. A coordinator is an agent that heads a sub-hierarchy. Leaf-nodes are simply termed *agents* in this model.

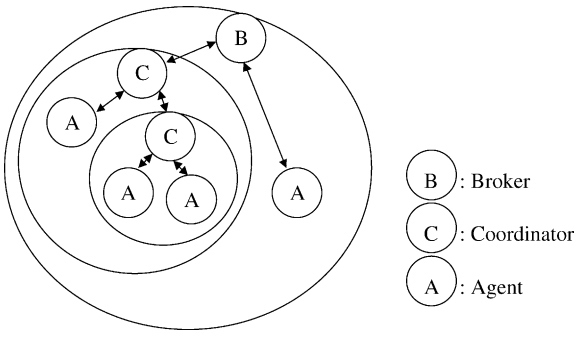

Fig. 1. Agent hierarchy.

Brokers, coordinators and agents are differentiated by their position in the hierarchy but not by their functionality. Each node has the same capabilities and priority for service and could perform services on behalf of any other node. This homogeneous hierarchy provides a high-level abstraction of a grid environment.

The agent hierarchy can represent an open and dynamic system. New agents can join the hierarchy or existing agents can leave the hierarchy as appropriate. The hierarchy exists only logically and each agent can contact others as long as it has their identities.

The hierarchical model partly addresses the issues of scalability. When the number of agents increases, the hierarchy may lead to many system activities being processed in a local domain. In this way the system may scale well and does not need to rely on one or a few central agents, which may otherwise become system bottlenecks.

Service is another important concept. An agent provides a simple and uniform abstraction of the functions (clients, services and go-betweens) in the grid management system. The service information provided at each local grid resource can be advertised throughout the hierarchy and agents can cooperate with each other to discover available resources. These are introduced in detail in Section 4.

2.3. Performance prediction

Performance prediction for parallel programs plays a key role for agents to perform resource scheduling and load balancing. Agents are integrated with existing PACE application performance prediction capabilities.

The PACE evaluation engine is the kernel of the PACE toolkit. The evaluation engine combines the PACE resource model (including performance related information of the hardware on which the parallel program will be executed) and application model (including all performance related information of the parallel program, e.g. MPI or PVM programs) at run time to produce evaluation results, e.g. estimation of execution time. Agents are equipped with the PACE evaluation engine and use predictive application performance data for scheduling. Detailed introduction to the PACE toolkit is out of the scope of this paper but the use of PACE performance prediction for both local grid and global grid load balancing is described below in Sections 3 and 4, respectively.

3. Local grid load balancing

In this section, a local grid resource is considered to be a cluster of workstations or a multiprocessor, which is abstracted uniformly as peer-to-peer networked hosts. Two algorithms are considered in the local management layer of each agent to perform local grid load balancing.

3.1. First-come-first-served algorithm

Consider a grid resource with n hosts where each host H_i has its own type ty_i. A PACE resource model can be used to describe the performance information of this host.

Let m be the number of considered tasks T. The arrival time of each task T_j is t_j. A PACE application model tm_j can be used to describe the application level performance information of each task. The user requirement of deadline for the task execution is represented as tr_j. Each task T_j also has two scheduled attributes – a start time ts_j and an end time te_j.

MT_j is the set of hosts that are allocated to task T_j. M then is a 2D array, which describes the mapping relationships between hosts and tasks using Boolean values.

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

$$M_{ij} = \begin{cases} 1, & \text{if } H_i \in MT_j \\ 0, & \text{if } H_i \notin MT_j \end{cases} \tag{2}$$

The PACE evaluation engine can produce performance prediction information based on the application model tm_j and resource models ty. An appropriate subset of hosts $\overline{H}$ (note that $\overline{H}$ cannot be an empty set Φ) can be selected, and this is evaluated and expressed as follows:

$$\forall \overline{H} \subseteq H, \ \ \overline{H} \neq \Phi, \ \ \overline{\mathrm{ty}} \subseteq \mathrm{ty}, \ \ \overline{\mathrm{ty}} \neq \Phi,$$
$$\overline{\mathrm{texe}}_j = \mathrm{eval}(\overline{\mathrm{ty}}, \mathrm{tm}_j) \tag{3}$$

The function of the agent local management is to find the earliest possible time for each task to complete, adhering to the sequence of the task arrivals.

$$\mathrm{te}_j = \min_{\forall \overline{H} \subseteq H, \overline{H} \neq \Phi} (\overline{\mathrm{te}}_j) \tag{4}$$

A task has the possibility of being allocated to any selection of hosts. The agent should consider all these possibilities and choose the earliest task end time. In any of these situations, the end time is equal to the earliest possible start time plus the execution time, which is described as follows:

$$\overline{\mathrm{te}}_j = \overline{\mathrm{ts}}_j + \overline{\mathrm{texe}}_j \tag{5}$$

The earliest possible start time for the task T_j on a selection of hosts is the latest free time of all the selected hosts if there are still tasks running on the selected hosts. If there is no task running on the selected hosts when the task T_j arrives at time t_j, T_j can be executed on these hosts immediately. These are expressed as follows:

$$\overline{\mathrm{ts}}_j = \max\left(t_j, \max_{\forall i, H_i \in \overline{H}}(\mathrm{td}_{ij})\right) \tag{6}$$

where td_{ij} is the latest free time of host H_i at the time t_j. This equals the maximum end times of tasks that are allocated to the host H_i before the task T_j arrives:

$$\mathrm{td}_{ij} = \max_{\forall p<j, M_{ip}=1}(\mathrm{te}_p) \tag{7}$$

In summary, te_j can be calculated as follows:

$$\mathrm{te}_j = \min_{\forall \overline{H} \subseteq H, \overline{H} \neq \Phi} \left(\max\left(t_j, \max_{\forall i, H_i \in \overline{H}}\left(\max_{\forall p<j, M_{ip}=1}(\mathrm{te}_p)\right)\right) + \overline{\mathrm{texe}}_j\right) \tag{8}$$

It is not necessarily the case that scheduling all hosts to a task will achieve higher performance. On the one hand, the start time of task execution may be earlier if only a number of processors are selected; on the other hand, with some tasks, execution time may become longer if too many hosts are allocated.

The complexity of the above algorithm is determined by the number of possible host selections. It is clear that if the number of hosts of a grid resource increases, the scheduling complexity will increase exponentially. This is based on a first-come-first-served policy that means the sequence of the task arrivals determines that of task executions. Reordering the task set may optimise the task execution further, but will increase the algorithm complexity. This is addressed using an iterative heuristic algorithm described below.

3.2. Genetic algorithm

When tasks can be reordered, the scheduling objective is also changed. Rather than looking for an earliest completion time for each task individually, the scheduling algorithm described in this section focuses on the makespan ω, which represents the latest completion time when all the tasks are considered together and is subsequently defined as:

$$\omega = \max_{1 \leq j \leq m}\{\mathrm{te}_j\} \tag{9}$$

The goal is to minimise function (9), at the same time $\forall j$, $\mathrm{te}_j \leq \mathrm{tr}_j$ should also be satisfied as far as possible. In order to obtain near optimal solutions to this combinatorial optimisation problem, the approach taken in this work is to find schedules that meet the above criteria through the use of an iterative heuristic method – in this case a genetic algorithm (GA). The process involves building a set of schedules and identifying solutions that have desirable characteristics. These are then carried into the next generation.

The technique requires a coding scheme that can represent all legitimate solutions to the optimisation problem. Any possible solution is uniquely represented by a particular string, and strings are manipulated in various ways until the algorithm converges on a near optimal solution. In order for this manipulation to proceed in the correct direction, a method of prescribing a quality value (or *fitness*) to each solution string is required. The algorithm for providing this value is called the fitness function f_v.

The coding scheme we have developed for this problem consists of two parts: an ordering part S_k, which specifies the order in which the tasks are to be executed, and a mapping part M_{ijk}, which specifies the host allocation to each task. Let k be the number of schedules in the scheduling set. The ordering of M_{ijk} is commensurate with the task order.

A combined cost function is used which considers makespan, idle time and deadline. It is straightforward to calculate the makespan, ω_k, of the schedule k represented by S_k and M_{ijk}. Let T_{jk} be the reordered task set

according to the ordering part of the coding scheme, S_k.

$$\mathrm{ts}_{jk} = \max_{\forall i, M_{ijk}=1} \left(\max_{\forall p<j, M_{ipk}=1} (\mathrm{te}_{pk}) \right) \tag{10}$$

$$\mathrm{te}_{jk} = \mathrm{ts}_{jk} + \mathrm{texe}_{jk} \tag{11}$$

$$\omega_k = \max_{1\le j\le m} \{\mathrm{te}_{jk}\} \tag{12}$$

The timing calculation described above is similar to that given in the function (8). One difference is that since all of the tasks are considered together, the order is defined according to S_k instead of the task arrival time t_j. So the consideration of t_j is not necessary in (10) as opposed to the function (6). Another aspect is that the host selection is defined using M_{ijk} and the PACE evaluation result texe_{jk} is calculated directly using corresponding resource models, while in the function (8), different possible host selections $\overline{H}$ have all to be considered and compared.

The nature of the idle time should also be taken into account. This is represented using the average idle time of all hosts φ_k.

$$\varphi_k = \max_{1\le j\le m} \{\mathrm{te}_{jk}\} - \min_{1\le j\le m} \{\mathrm{ts}_{jk}\} - \frac{\sum_{j=1}^{m}\sum_{i=1}^{n} M_{ijk}(\mathrm{te}_{jk} - \mathrm{ts}_{jk})}{n} \tag{13}$$

The contract penalty θ_k is derived from the expected deadline times tr and task completion time te.

$$\theta_k = \frac{\sum_{j=1}^{m}(\mathrm{te}_{jk} - \mathrm{tr}_j)}{m} \tag{14}$$

The cost value for the schedule k, represented by S_k and M_{ijk}, is derived from these metrics and their impact predetermined by:

$$f_c^k = \frac{W^m\omega_k + W^i\varphi_k + W^c\theta_k}{W^m + W^i + W^c} \tag{15}$$

The genetic algorithm utilises a fixed population size and stochastic remainder selection. Specialised crossover and mutation functions are developed for use with the two-part coding scheme. The crossover function first splices the two ordering strings at a random location, and then reorders the pairs to produce legitimate solutions. The mapping parts are crossed over by first reordering them to be consistent with the new task order, and then performing a single-point (binary) crossover. The reordering is necessary to preserve the node mapping associated with a particular task from one generation to the next. The mutation stage is also two-part, with a switching operator randomly applied to the ordering parts, and a random bit-flip applied to the mapping parts.

In the actual agent implementation using the above algorithm, the system dynamism must be considered. One advantage of the iterative algorithm described in this section is that it is an evolutionary process and is therefore able to absorb system changes such as the addition or deletion of tasks, or changes in the number of hosts available in the grid resource.

The two scheduling algorithms are both implemented and can be switched from one to another in the agent. The algorithms provide a fine-grained solution to dynamic task scheduling and load balancing across multiple hosts of a local grid resource. However, the same methodology cannot be applied directly to a large-scale grid environment, since the algorithms do not scale to thousands of hosts and tasks. An additional mechanism is required for multiple agents to work together and achieve global grid load balancing.

4. Global grid load balancing

In this work, a grid is a collection of multiple local grid resources that are distributed geographically in a wide area. The problem that is addressed in this section is the discovery of available grid resources that provide the optimum execution performance for globally grid-submitted tasks. The service discovery process indirectly results in a load-balancing effect across multiple grid resources.

4.1. Service advertisement and discovery

An agent takes its local grid resource as one of its capabilities. An agent can also receive many service advertisements from nearby agents and store this information in its coordination layer as its own knowledge. All of the service information are organised into Agent Capability Tables (ACTs). An agent can choose to maintain different kinds of ACTs according to different sources of service information. These include:

- *T_ACT*: In the coordination layer of each agent, T_ACT is used to record service information of the local grid resource. The local management layer is responsible for collecting this information and reporting it to the coordination layer.
- *L_ACT*: Each agent can have one L_ACT to record the service information received from its lower agents in the hierarchy. The services recorded in L_ACT are provided by grid resources in its local scope.
- *G_ACT*: The G_ACT in an agent is actually a record of the service information received from its upper agent in the hierarchy. The service information recorded in G_ACT is provided by the agents, which have the same upper agent as the agent itself.

There are basically two ways to maintain the contents of ACTs in an agent: data-pull and data-push, each of which has two approaches: periodic and event-driven.

- *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 hierarchy periodically or when the service information is changed.

Apart from service advertisement, another important process among agents is service discovery. Discovering available services is also a cooperative activity. Within each agent, its own service provided by the local grid resource is evaluated first. If the requirement can be met locally, the discovery ends successfully. Otherwise service information in both L_ACT and G_ACT is evaluated and the request dispatched to the agent, which is able to provide the best requirement/resource match. If no service can meet the requirement, the request is submitted to the upper agent. When the head of the hierarchy is reached and the available service is still not found, the discovery terminates unsuccessfully.

While the process of service advertisement and discovery is not motivated by grid scheduling and load balancing, it can result in an indirect coarse-grained load-balancing effect. A task tends to be dispatched to a grid resource that has less workload and can meet the application execution deadline. The discovery process does not aim to find the best service for each request, but endeavours to find an available service provided by a neighbouring agent. While this may decrease the load-balancing effect, the trade-off is reasonable as grid users prefers to find a satisfactory resource as fast and as local as possible.

The advertisement and discovery mechanism also allows possible system scalability. Most requests are processed in a local domain and need not to be submitted to a wider area. Both advertisement and discovery requests are processed between neighbouring agents and the system has no central structure, which otherwise might act as a potential bottleneck.

4.2. System implementation

Agents are implemented using Java and data are represented in an XML format. An agent is responsible for collecting service information of the local grid resource. An example of this service information can be found below.

```
<agentgrid type=''service''>
  <address>gem.dcs.warwick.ac.uk</address>
  <port>1000</port>
  <type>SunUltra10</type>
  <nproc>16</nproc>
  <environment>mpi</environment>
  <environment>pvm</environment>
  <environment>test</environment>
  <freetime>Nov 1504:43:102001</freetime>
</agentgrid>
```

The agent identity is provided by a tuple of the *address* and *port* used to initiate communication. The hardware model and the number of processors are also provided. The example specifies a single cluster, in this case a cluster of 16 SunUltra10 workstations. To simplify the problem, the hosts within each grid resource are configured to be homogeneous. The application execution environments that are supported by the current agent implementation include MPI, PVM, and a *test* mode that is designed for the experiments described in this work. Under *test* mode, tasks are not actually executed and predictive application execution times are scheduled and assumed to be accurate. The latest scheduling makespan ω indicates the earliest (approximate) time that corresponding grid resource become available for more tasks. Due to the effect of load balancing, it is reasonable to assume that all of hosts within

a grid resource have approximately the same *freetime*. The agents use this item to estimate the workload of each grid resource and make decisions on where to dispatch incoming tasks. This item changes over time and must be frequently updated. Service advertisement is therefore important among the agents.

A portal has been developed which allows users to submit requests destined for the grid resources. An example request is given below.

```
<agentgrid type=''request''>
  <application>
    <name>sweep3d</name>
    <binary>
      <file>binary/sweep3d</file>
      <inputfile>input/input.50</inputfile>
    </binary>
    <performance>
      <datatype>pacemodel</datatype>
      <modelname>model/sweep3d</modelname>
    </performance>
  </application>
  <requirement>
    <environment>test</environment>
    <deadline>Nov 1504:43:17 2001</deadline>
  </requirement>
  <email>junwei</email>
</agentgrid>
```

A user is required to specify the details of the application, the requirements and contact information for each request. Application information includes binary executable files and also the corresponding PACE application performance model tm_r. In the current implementation we assume that both binary and model files are pre-compiled and available in all local file systems. In the requirements, both the application execution environment and the required deadline time tr_r should be specified. Currently the user's email address is used as the contact information.

Service discovery processes are triggered by the arrival of a request at an agent, where the kernel of this process is the matchmaking between service and request information. The match is straightforward whether an agent can provide the required application execution environment. The expected execution completion time for a given task on a given resource can be estimated using:

$$\mathrm{te}_r = \omega + \min_{\forall \overline{H} \subseteq H, \overline{H} \neq \Phi, \overline{\mathrm{ty}} \subseteq \mathrm{ty}, \overline{\mathrm{ty}} \neq \Phi} \{\mathrm{eval}(\overline{\mathrm{ty}}, \mathrm{tm}_r)\} \quad (16)$$

For a grid resource with homogeneous hosts, the PACE evaluation function is called n times. If $\mathrm{te}_r \leq \mathrm{tr}_r$, the resource is considered to be able to meet the required deadline. Otherwise, the resource is not considered available for the incoming task. This performance estimation of local grid resources at the global level is simple but efficient. However, when the task is dispatched to the corresponding agent, the actual situation may differ from the scenario considered in (16). The agent may change the task order and advance or postpone a specific task execution in order to balance the workload on different hosts, and in so doing maximise resource utilisation while maintaining the deadline contracts of each user.

Service discovery for a request within an agent involves multiple matchmaking processes. An agent always gives priority to the local grid resource. Only when the local resource is unavailable is the service information of other grid resources evaluated and the request dispatched to another agent. In order to measure the effect of this mechanism for grid scheduling and load balancing, several performance metrics are defined and many experiments are carried out.

5. Performance evaluation

Experiments are carried out for performance evaluation of grid load balancing using intelligent agents described in above sections. Performance metrics are predefined and experimental results are included in this section.

5.1. Performance metrics

There are a number of performance criteria that can be used to describe resource management and scheduling systems. What is considered as high performance depends on the system requirements. In this work, there are several common statistics that can be investigated quantitatively and used to characterise the effect of scheduling and load balancing.

5.1.1. Total application execution time

This defines the period of time t, when a set of m parallel tasks T are scheduled onto resources H with n hosts. Note that the host set H here is slightly different from that defined in Section 3.1 because it may include those either at multiple grid resources or within a single grid resource.

$$t = \max_{1\le j\le m}\{\mathrm{te}_j\} - \min_{1\le j\le m}\{\mathrm{ts}_j\} \tag{17}$$

5.1.2. Average advance time of application execution completion

This can be calculated directly using:

$$\varepsilon = \frac{\sum_{j=1}^{m}(\mathrm{tr}_j - \mathrm{te}_j)}{m} \tag{18}$$

which is negative when most deadlines fail.

5.1.3. Average resource utilisation rate

The resource utilisation rate υ_i of each host H_i is calculated as follows:

$$\upsilon_i = \frac{\sum_{\forall j, M_{ij}=1}(\mathrm{te}_j - \mathrm{ts}_j)}{t} \times 100\% \tag{19}$$

The average resource utilisation rate υ of all hosts H is:

$$\upsilon = \frac{\sum_{i=1}^{n}\upsilon_i}{n} \tag{20}$$

where υ is in the range 0–1.

5.1.4. Load-balancing level

The mean square deviation of υ_i is defined as:

$$d = \sqrt{\frac{\sum_{i=1}^{n}(\upsilon - \upsilon_i)^2}{n}} \tag{21}$$

and the relative deviation of d over υ that describes the load-balancing level of the system is:

$$\beta = \left(1 - \frac{d}{\upsilon}\right) \times 100\% \tag{22}$$

The most effective load balancing is achieved when d equals zero and β equals 100%. The four aspects of the system described above can be applied both to a grid resource or a grid environment that consists of multiple grid resources. These performance metrics are also interrelated. For example, if the workload is balanced across all the considered hosts, the resource utilisation rate is usually high and the tasks finish quickly. Another metrics that can only applied for measurement of grid agents is the number of network packets used for service advertisement and discovery.

5.2. Experimental design

The experimental system is configured with 12 agents, illustrated by the hierarchy shown in Fig. 2.

These agents are named $S_1, \ldots, S_{12}$ (for the sake of brevity) and represent heterogeneous hardware resources containing 16 hosts/processors per resource. As shown in Fig. 2, the resources range in their computational capabilities. The SGI multiprocessor is the most powerful, followed by the SunUltra10, 5, 1, and SPARCStation 2 in turn.

In the experimental system, each agent maintains a set of service information for the other agents in the system. Each agent pulls service information from its lower and upper agents in every 10 s. All of the agents employ identical strategies with the exception of the agent at the head of the hierarchy (S_1) that does not have an upper agent.

The applications used in the experiments include typical scientific computing programs. Each application has been modelled and evaluated using PACE. Different applications have different performance scenarios. During each experiment, requests for one of the test applications are sent at 1-s intervals to randomly selected agents. The required execution time deadline for the application is also selected randomly from a given domain. The request phase of each experiment lasts for 10 min during which 600 task execution requests are sent out to the agents.

While the experiments use the same resource configurations and application workloads described above, different combinations of local grid scheduling algorithms and global grid mechanisms are applied as shown in Table 1.

5.3. Experimental results

The experimental results are given in Table 2; this includes the three metrics applied to each agent and to all the grid resources in the system.

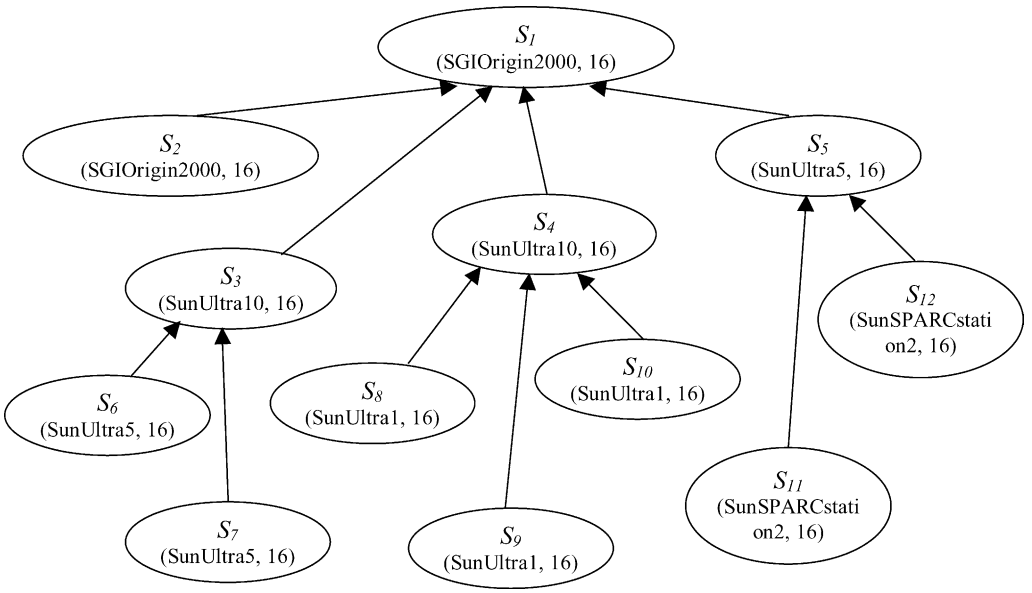

Fig. 2. Case study: agents and resources.

Table 1
Case study: experimental design

	Experiment number		
	1	2	3
First-come-first-served algorithm	√		
Iterative heuristic algorithm		√	√
Service advertisement and discovery			√

5.3.1. Experiment 1

In the first experiment, each agent is configured with the first-come-first-served algorithm for local grid resource scheduling. Agents are not organised for co-operation. The experimental scenario is visualised in Fig. 3.

The algorithm does not consider makespan, idle time or deadline. Each agent receives approximately

Table 2
Case study: experimental results

	Experiment number								
	1			2			3		
	ε (s)	υ (%)	β (%)	ε (s)	υ (%)	β (%)	ε (s)	υ (%)	β (%)
S_1	42	7	71	52	9	89	29	81	96
S_2	11	9	78	34	9	89	23	81	95
S_3	−135	13	62	23	13	92	24	77	87
S_4	−328	22	45	−30	28	96	44	82	94
S_5	−607	32	56	−492	58	95	38	82	94
S_6	−321	25	56	−123	29	90	42	78	92
S_7	−261	23	57	10	25	92	38	84	93
S_8	−695	33	52	−513	52	90	42	82	91
S_9	−806	45	58	−724	63	90	30	80	84
S_{10}	−405	28	61	−129	34	94	25	81	94
S_{11}	−1095	44	50	−816	73	92	35	75	89
S_{12}	−859	41	46	−550	67	91	26	78	90
Total	−475	26	31	−295	38	42	32	80	90

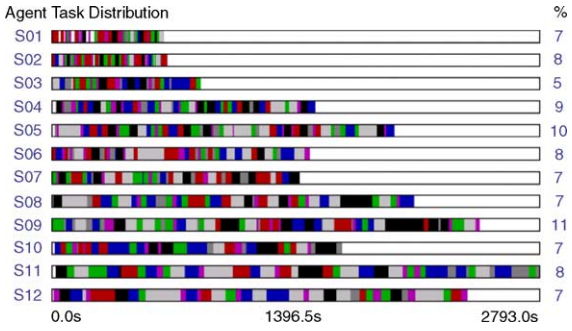

Fig. 3. Experimental scenario I.

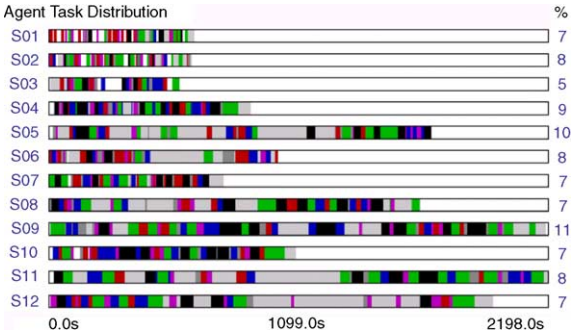

Fig. 4. Experimental scenario II.

50 task requests on average, which results in only the powerful platforms (SGI multiprocessors S_1 and S_2) meeting the requirements. The slower machines including the Sun SPARCStations clusters S_{11} and S_{12} impose serious delays in task execution with long task queues (see Fig. 3). The total task execution time is about 46 min. The overall average delay for task execution is approximately 8 min. It is apparent that the high performance platforms are not utilised effectively, and the lack of proper scheduling overloads clusters like S_{11} that is only 44% utilised. The average utilisation of grid resources is only 26%. The workload for each host in each grid resource is also unbalanced. For example, the load-balancing level of S_{12} is as low as 46%. The overall grid workload is also unbalanced at 31%.

5.3.2. Experiment 2

In experiment 2, the iterative heuristic algorithm is used in place of the first-come-first-serve algorithm although no higher level agent cooperation mechanism is applied. The experimental scenario is visualised in Fig. 4.

The algorithm aims to minimise makespan and idle time, while meeting deadlines. Compared to those of experiment 1, almost all metrics are improved. Task executions are completed earlier. The total task execution time is improved from 46 to 36 min and the average task execution delay is reduced to approximately 5 min. However, resources such as S_{11} and S_{12} remain overloaded and the GA scheduling is not able to find solutions that satisfy all the deadlines. Generally, resources are better utilised as a result of the better scheduling, such as the use of S_{11} that increases from 44 to 73%. The overall average utilisation also improves from 26 to 38%. While load balancing on each grid resources is significantly improved, the lack of any higher level load-balancing mechanism results in a slightly improved overall grid load balancing to 42% (as opposed to 31% in experiment 1).

5.3.3. Experiment 3

In experiment 3, the service advertisement and discovery mechanism is enabled for high-level load balancing. The experimental scenario is visualised in Fig. 5.

Service discovery results in a new distribution of requests to the agents, where the more powerful platform receives more requests. As shown in Fig. 5, powerful platform like S_1 receives 16% of tasks, which is four times of tasks received by relatively slow platform S_{11}. The total task execution time is also dramatically decreased to 11 min. As a result, the majority of task execution requirements can be met and all grid resources are well utilised (80% on average). The load balancing of the overall grid is significantly improved from 42% (in experiment 2) to 90%. The load balancing on

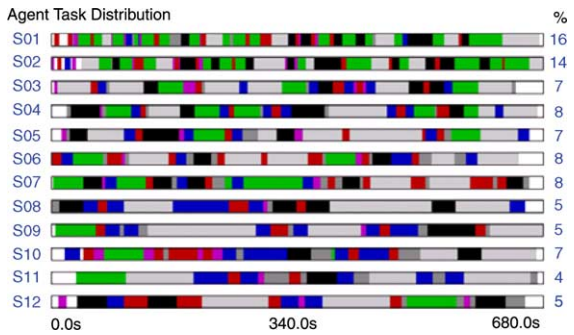

Fig. 5. Experimental scenario III.

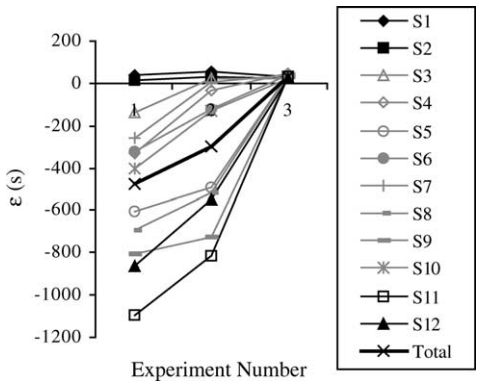

Fig. 6. Case study: trends I for experimental results on advance times of application execution completion ε.

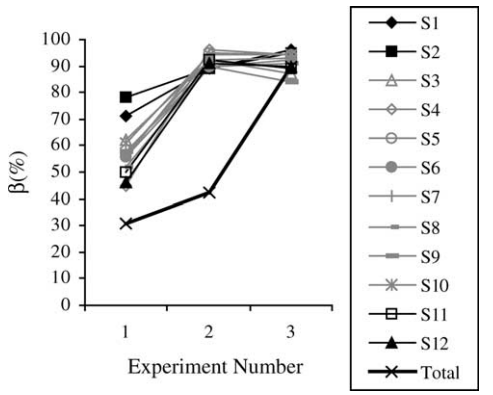

Fig. 8. Case study: trends III for experimental results on load-balancing level β.

resources such as S_1 and S_2 are only marginally improved by the GA scheduling when the workload is higher. None of other agents show an improvement in local grid load balancing.

Experimental results in Table 2 are also illustrated in Figs. 6–8, showing the effect on the performance metrics given in Section 5.1. The curves indicate that different platforms exhibit different trends when agents are configured with more scheduling and load-balancing mechanisms. Among these the curves for S_1, S_2 (which are the most powerful) and S_{11}, S_{12} (which are the least powerful) are representative and are therefore emphasised, while others are indicated using grey lines. The curve for the overall grid is illustrated using a bold line.

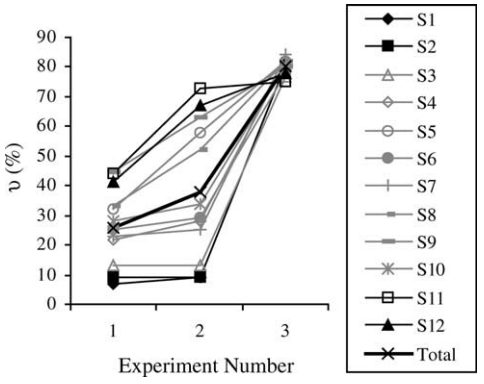

Fig. 7. Case study: trends II for experimental results on resource utilisation rate υ.

5.3.4. Application execution

In Fig. 6, it is apparent that both the GA scheduling and the service discovery mechanism contribute to improving the application execution completion.

The curve implies that the more a resource is loaded the more significant the effect is. For example, S_1 and S_2 are not overloaded during the three experiments, and therefore the value of ε only changes slightly. S_{11} and S_{12} are heavily overloaded during the experiments 1 and 2, and therefore the improvement of ε in the experiments 2 and 3 is more significant. The situations of S_3, ..., S_{10} are distributed between these two extremes. The curve for the overall grid provides an average estimation for all situations, which indicates that the service discovery mechanism contributes more towards the improvement in application executions than GA scheduling.

5.3.5. Resource utilisation

The curves in Fig. 7 illustrate similar trends to those of Fig. 6. S_1, S_2 and S_{11}, S_{12} still represent the two extreme situations between which the other platforms are distributed.

The curve for the overall grid indicates that the service discovery mechanism contributes more to maximising resource utilisation. However, overloaded platforms like S_{11} and S_{12} benefit mainly from the GA scheduling, which is more effective at load balancing when the workload is high; lightly loaded platforms like S_1 and S_2 chiefly benefit from the service discovery mechanism, which can dispatch more tasks to them.

5.3.6. Load balancing

Curves in Fig. 8 demonstrate that local and global grid load balancing are achieved in different ways.

While S_1, S_2 and S_{11}, S_{12} are two representative situations, the global situation is not simply an average of local trends as those illustrated in Figs. 6 and 7. In the second experiment, when the GA scheduling is enabled, the load balancing of hosts or processors within a local grid resource are significantly improved. In the third experiment, when the service discovery mechanism is enabled, the overall grid load balancing is improved dramatically. It is clear that the GA scheduling contributes more to local grid load balancing and the service discovery mechanism contributes more to global grid load balancing. The coupling of both as described in this work is therefore a good choice to achieve load balancing at both local and global grid levels.

5.4. Agent performance

Additional experiments are carried out to compare the performance of grid agents when different service advertisement and discovery strategies are applied. These are introduced briefly in this section.

A centralised controlling mechanism is designed for the agents. Each agent is assumed to have the preknowledge of any other agents. Each time an agent receives a task execution request, it will contact all of the other agents for quoting of completion time. The best bid is chosen and the request is dispatched to the available grid resource directly in one step. This is actually an event-driven data-pull strategy, which means that full advertisement results in no necessary discovery.

The service advertisement strategy used in the last section is periodic data-pull, where service information is only transferred among neighbouring agents. This results that service discovery has also to be processed step by step. This distributed strategy means that not full advertisement results in necessary discovery steps. The experimental results introduced below indicate that balancing the overhead for advertisement and discovery in this way can lead to a better agent performance.

The details of the experimental design are not included, though actually very similar to that introduced in Section 5.2. One difference is that in these experiments, the number of grid agents is changed to enable the system scalability to be investigated.

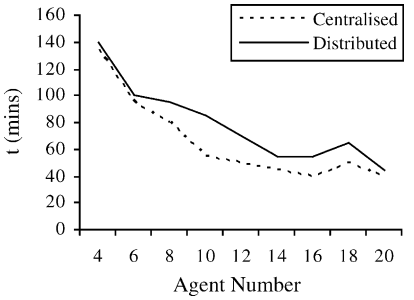

Fig. 9. Comparison of total application execution time between the centralised and distributed strategies.

5.4.1. Total application execution time

Fig. 9 provides a comparison of total application execution time for the two strategies.

The total task execution time decreases when the number of agents and grid resources increases. It is clear that the centralised strategy leads to a bit better load-balancing results, since tasks finish in a less time under the centralised control. This is more obvious when the number of the agents increases.

It is reasonable that a centralised strategy can achieve a better scheduling, because full service advertisement leads to full knowledge on the performance of all grid resources. However, under a distributed mechanism, each agent has only up-to-date information on its neighbouring agents, which limit the scheduling effect.

5.4.2. Average advance time of application execution completion

Similar comparison for the two strategies is included in Fig. 10 in terms of the average advance time of application execution time.

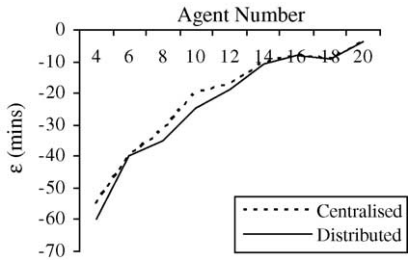

Fig. 10. Comparison of average advance time of application execution completion between the centralised and distributed strategies.

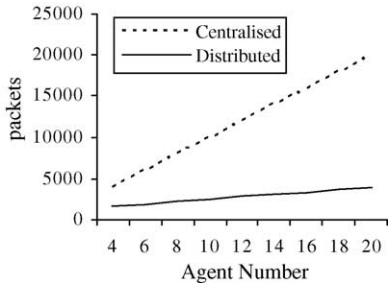

Fig. 11. Comparison of network packets between the centralised and distributed strategies.

Tasks are executed quicker when the number of agents increases. It is clear that the centralised strategy leads to a bit better result again. The reason is similar to that described in the last section. The result values are negative since the workload of these experiments is quite heavy and grid resources cannot meet the deadline requirements of task execution averagely.

5.4.3. Network packets

A different result is included in Fig. 11 that provides a comparison of the network packets involved during the experiments of the two strategies.

The number of network messages used for service advertisement and discovery increases linearly with the number of agents. It is clear that the distributed strategy significantly decreases the amount of network traffic. The strategy of only passing messages among neighbouring agents improves the system scalability as the agent number increases.

6. Related work

In this work, local grid load balancing is performed in each agent using AI scheduling algorithms. The on-the-fly use of predictive performance data for scheduling described in this work is similar to that of AppLeS [5], Ninf [30] and Nimrod [2]. While AppLeS and Ninf management and scheduling are also based on performance evaluation techniques, they utilise the NWS [39] resource monitoring service. Nimrod has a number of similarities to this work, including a parametric engine and heuristic algorithms [1] for scheduling jobs. There are also many job scheduling systems, such as Condor [28], EASY [27], Maui [25], LSF [40] and PBS [24]. Most of these support batch queuing using the FCFS algorithm. The main advantage of GA scheduling used in this work for job scheduling is the quality of service and multiple performance metrics support.

This work also focuses on the cooperation of local grid and global grid levels of management and scheduling. OGSA and its implementation, the Globus toolkit [19], is the standard for grid service and application development, which is based on web services protocols and standards [31]. Some existing systems use the Globus toolkit to integrate with the grid computing environment, including Condor-G [22], Nimrod/G [3], though a centralised control structure is applied in both implementations. Another grid computing infrastructure, Legion [23], is developed using an object-oriented methodology that provides similar functionalities to the Globus. In this work, a multi-agent approach is considered. Agents are used to control the query process and to make resource discovery decisions based on internal logic rather than relying on a fixed-function query engine.

Agent-based grid management is also used in JAMM [7,38] and NetSolve [17,18], where a centralised broker/agents architecture is developed. In this work, agents perform peer-to-peer service advertisement and discovery to achieve global grid load balancing. Compared with another “Agent Grid” work described in [33], rather than using a collection of many predefined specialised agents, grid load balancing in this work uses a hierarchy of homogeneous agents that can be reconfigured with different roles at running time. While there are also several other related projects that have a focus on agent-based grid computing [29,34,37], the emphases of these works are quite different. In this work, performance for grid load balancing is investigated in a quantitative way that cannot found in any other work.

There are many other enterprise computing and middleware technologies that are being adopted for grid management, such as CORBA [35] and Jini [4]. Compared with these methods, the most important advantage of an agent-based approach is that it can provide a clear high-level abstraction of the grid environment that is extensible and compatible for integration of future grid services and toolkits.

7. Conclusions

This work addresses grid load-balancing issues using a combination of intelligent agents and multi-agent approaches. For local grid load balancing, the iterative heuristic algorithm is more efficient than the first-come-first-served algorithm. For global grid load balancing, a peer-to-peer service advertisement and discovery technique is shown to be effective. The use of a distributed agent strategy can reduce the network overhead significantly and allow the system to scale well rather than using a centralised control, as well as achieving a reasonable good resource utilisation and meeting application execution deadlines.

Further experiments will be carried out using the grid testbed being built at Warwick. Since large-scale deployments of developing systems are problematic, a grid modelling and simulation environment is under development to enable performance and scalability of the agent system to be investigated when thousands of grid resources and agents are involved.

The next generation grid computing environment must be intelligent and autonomous to meet requirements of self management. Related research topics include semantic grids [41] and knowledge grids [42]. The agent-based approach described in this work is an initial attempt towards a distributed framework for building such an intelligent grid environment. Future work includes the extension of the agent framework with new features, e.g. automatic QoS negotiation, self-organising coordination, semantic integration, knowledge-based reasoning and ontology-based service brokering.

References

[1] A. Abraham, R. Buyya, B. Nath, Nature's heuristics for scheduling jobs on computational grids, in: Proceedings of the ADCOM'00, Cochin, India, 2000.
[2] D. Abramson, R. Sosic, J. Giddy, B. Hall, Nimrod: a tool for performing parameterized simulations using distributed workstations, in: Proceedings of the HPDC'95, Pentagon City, VA, USA, 1995.
[3] D. Abramson, J. Giddy, L. Kotler, High performance parametric modelling with Nimrod/G: killer application for the global grid, in: Proceedings of the IPDPS'00, Cancun, Mexico, 2000.
[4] K. Amold, B. O'Sullivan, R. Scheifer, J. Waldo, A. Woolrath, The Jini™ Specification, Addison Wesley, 1999.
[5] F. Berman, R. Wolski, S. Figueira, J. Schopf, G. Shao, Application-level scheduling on distributed heterogeneous networks, in: Proceedings of the Supercomputing'96, Pittsburgh, PA, USA, 1996.
[6] F. Berman, A.J.G. Hey, G. Fox, Grid Computing: Making The Global Infrastructure a Reality, John Wiley & Sons, 2003.
[7] C. Brooks, B. Tierney, W. Johnston, JAVA agents for distributed system management, LBNL Report, 1997.
[8] J. Cao, D.J. Kerbyson, E. Papaefstathiou, G.R. Nudd, Performance modelling of parallel and distributed computing using PACE, in: Proceedings of the IPCCC'00, Phoenix, AZ, USA, 2000, pp. 485–492.
[9] J. Cao, D.J. Kerbyson, G.R. Nudd, Dynamic application integration using agent-based operational administration, in: Proceedings of the PAAM'00, Manchester, UK, 2000, pp. 393–396.
[10] J. Cao, D.J. Kerbyson, G.R. Nudd, High performance service discovery in large-scale multi-agent and mobile-agent systems, Int. J. Software Eng. Knowl. Eng. 11 (5) (2001) 621–641.
[11] J. Cao, D.J. Kerbyson, G.R. Nudd, Use of agent-based service discovery for resource management in metacomputing environment, in: Proceedings of the Euro-Par'01, Lecture Notes on Computer Science, vol. 2150, Springer, Berlin, 2001, pp. 882–886.
[12] J. Cao, D.J. Kerbyson, G.R. Nudd, Performance evaluation of an agent-based resource management infrastructure for grid computing, in: Proceedings of the CCGrid'01, Brisbane, Australia, 2001, pp. 311–318.
[13] J. Cao, D.P. Spooner, J.D. Turner, S.A. Jarvis, D.J. Kerbyson, S. Saini, G.R. Nudd, Agent-based resource management for grid computing, in: Proceedings of the AgentGrid'02, Berlin, Germany, 2002, pp. 350–351.
[14] J. Cao, S.A. Jarvis, S. Saini, D.J. Kerbyson, G.R. Nudd, ARMS: an agent-based resource management system for grid computing, Scientific Programming (Special Issue on Grid Computing) 10 (2) (2002) 135–148.
[15] J. Cao, S.A. Jarvis, D.P. Spooner, J.D. Turner, D.J. Kerbyson, G.R. Nudd, Performance prediction technology for agent-based resource management in grid environments, in: Proceedings of the HCW'02, Fort Lauderdale, FL, USA, 2002.
[16] J. Cao, D.P. Spooner, S.A. Jarvis, S. Saini, G.R. Nudd, Agent-based grid load balancing using performance-driven task scheduling, in: Proceedings of the IPDPS'03, Nice, France, 2003.
[17] H. Casanova, J. Dongarra, Using agent-based software for scientific computing in the NetSolve system, Parallel Comput. 24 (12–13) (1998) 1777–1790.
[18] H. Casanova, J. Dongarra, Applying NetSolve's network-enabled server, IEEE Comput. Sci. Eng. 5 (3) (1998) 57–67.
[19] I. Foster, C. Kesselman, Globus: a metacomputing infrastructure toolkit, Int. J. High Perform. Comput. Appl. 2 (1997) 115–128.
[20] I. Foster, C. Kesselman, The GRID: Blueprint for a New Computing Infrastructure, Morgan-Kaufmann, 1998.
[21] I. Foster, C. Kesselman, J.M. Nick, S. Tuecke, Grid services for distributed system integration, IEEE Comput. 35 (6) (2002) 37–46.

[22] J. Frey, T. Tannenbaum, M. Livny, I. Foster, S. Tuecke, Condor-G: a computation management agent for multi-institutional grids, Cluster Comput. 5 (3) (2002) 237–246.
[23] A. Grimshaw, W.A. Wulf, Legion team, The Legion vision of a worldwide virtual computer, Commun. ACM 40 (1) (1997) 39–45.
[24] R.L. Henderson, Job scheduling under the Portable Batch System, in: Proceedings of the JSSPP'95, Lecture Notes in Computer Science, vol. 949, Springer, Berlin, 1995, pp. 279–294.
[25] D. Jackson, Q. Snell, M. Clement, Core algorithms of the Maui scheduler, in: Proceedings of the JSSPP'01, Lecture Notes Computer Science, vol. 2221, Springer, Berlin, 2001, pp. 87–102.
[26] N.R. Jennings, M.J. Wooldridge (Eds.), Agent Technology: Foundations, Applications, and Markets, Springer, Berlin, 1998.
[27] D. Lifka, The ANL/IBM SP scheduling system, in: Proceedings of the JSSPP'01, Lecture Notes Computer Science, vol. 2221, Springer, Berlin, 2001, pp. 187–191.
[28] M. Litzkow, M. Livny, M. Mutka, Condor – a hunter of idle workstations, in: Proceedings of the ICDCS'88, San Jose, CA, USA, 1988, pp. 104–111.
[29] L. Moreau, Agents for the grid: a comparison for web services (part 1: the transport layer), in: Proceedings of the CCGrid'02, Berlin, Germany, 2002, pp. 220–228.
[30] H. Nakada, M. Sato, S. Sekiguchi, Design and implementations of Ninf: towards a global computing infrastructure, Future Gen. Comput. Syst. 5–6 (1999) 649–658.
[31] E. Newcomer, Understanding Web Services: XML, WSDL, SOAP, and UDDI, Addison Wesley, 2002.
[32] G.R. Nudd, D.J. Kerbyson, E. Papaefstathiou, S.C. Perry, J.S. Harper, D.V. Wilcox, PACE – a toolset for the performance prediction of parallel and distributed systems, Int. J. High Perform. Comput. Appl. 14 (3) (2000) 228–251.
[33] O.F. Rana, D.W. Walker, The agent grid: agent-based resource integration in PSEs, in: Proceedings of the IMACS'00, Lausanne, Switzerland, 2000.
[34] W. Shen, Y. Li, H. Ghenniwa, C. Wang, Adaptive negotiation for agent-based grid computing, in: Proceedings of the Agentcities/AAMAS'02, Bologna, Italy, 2002, pp. 32–36.
[35] D. Slama, J. Garbis, P. Russell, Enterprise Corba, Prentice Hall, 1999.
[36] R. Stevens, P. Woodward, T. DeFanti, C. Catlett, From the I-WAY to the national technology grid, Commun. ACM 40 (11) (1997) 50–60.
[37] C. Thompson, Characterizing the agent grid, Technical Report, Object Services and Consulting Inc., 1998, http://www.objs.com/agility/tech-reports/9812-grid.html.
[38] B. Tierney, W. Johnston, J. Lee, M. Thompson, A data intensive distributed computing architecture for grid applications, Future Gen. Comput. Syst. 16 (5) (2000) 473–481.
[39] R. Wolski, N.T. Spring, J. Hayes, The network weather service: a distributed resource performance forecasting service for metacomputing, Future Gen. Comput. Syst. 15 (5–6) (1999) 757–768.
[40] S. Zhou, LSF: load sharing in large-scale heterogeneous distributed systems, in: Proceedings of the Cluster Computing, 1992.
[41] H. Zhuge, Semantics, resource and grid, Future Gen. Comput. Syst. 20 (1) (2004) 1–5.
[42] H. Zhuge, China's e-science knowledge grid environment, IEEE Intell. Syst. 19 (1) (2004) 13–17.

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

Daniel P. Spooner is a newly appointed Lecturer and a member of the High Performance System Group at the University of Warwick. He has 15 referred publications on the generation and application of analytical performance models to Grid computing systems. He has worked at the Performance and Architectures Laboratory at the Los Alamos National Laboratory on performance modelling tools, and is currently involved in an e-Science programme to develop performance-aware middleware services.

Stephen A. Jarvis is a Senior Lecturer in the High Performance System Group at the University of Warwick. He has authored over 50 referred publications (including three books) in the area of software and performance evaluation. While previously at the Oxford University Computing Laboratory, he worked on performance tools for a number of different programming paradigms. He has close research links with IBM, including current projects with IBM's TJ Watson Research Center in New York and with their development centre at Hursley Park in the UK. Dr Jarvis sits on a number of international programme committees for high-performance computing, autonomic computing and active middleware; he is also the Manager of the Midlands e-Science Technical Forum on Grid Technologies.

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