In Proceedings of 11th IEEE Heterogeneous Computing Workshop (HCW 2002), in conjunction with 16th International Parallel & Distributed Processing Symposium (IPDPS 2002), Fort Lauderdale, Florida, USA, April 2002, 14 pages.

Performance Prediction Technology for Agent-based Resource Management in Grid Environments

Junwei Cao, Stephen A. Jarvis, Daniel P. Spooner,
James D. Turner, Darren J. Kerbyson* and Graham R. Nudd,
High Performance Systems Group, University of Warwick, Coventry, UK
*Modelling, Algorithms, and Informatics Group, Los Alamos National Laboratory, USA
Email address for correspondence: junwei@dcs.warwick.ac.uk

Abstract

Resource management constitutes an important infrastructural component of a computational grid environment. The aim of grid resource management is to efficiently schedule applications over the available resources provided by the supporting grid architecture. Such goals within the high performance community rely, in part, on accurate performance prediction capabilities.

This paper introduces a resource management infrastructure for grid computing environments. The technique couples application performance prediction with a hierarchical multi-agent system. An initial system implementation utilises the performance prediction capabilities of the PACE toolkit to provide quantitative data regarding the performance of complex applications running on local grid resources. The validation results show that a high level of accuracy can be obtained, that cross-platform comparisons can be easily undertaken, and that the estimates can be evaluated rapidly.

A hierarchy of homogeneous agents are used to provide a scalable and adaptable abstraction of the grid system architecture. An agent is a representative of a local grid resource and is considered to be both a service provider and a service requestor. Agents are organised into a hierarchy and cooperate to provide service advertisement and discovery. A performance monitor and advisor has been developed to optimise the performance of the agent system. A case study with corresponding experimental results are included to demonstrate the efficiency of the resource management and scheduling system.

The main features of the system include: hard quality of service support using PACE performance prediction capabilities; agent-based dynamic resource advertisement and discovery capabilities; simulation-based quantitative grid performance analysis and user-oriented scheduling of local grid resources.

1. Introduction

It is anticipated that grid computing will deliver high performance computing capabilities and 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, which must efficiently map applications to available resources within a grid environment [20]. Such goals within the high performance community will rely on accurate performance prediction capabilities.

An existing toolkit, known as PACE (Performance Analysis and Characterisation Environment) [23], is used to provide quantitative predictive data concerning the performance of complex applications running on a local grid resource. The framework of PACE is based on a layered methodology that separates software and hardware system components through the use of parallelisation templates. The PACE performance prediction capabilities have been validated using an ASCI (Accelerated Strategic Computing Initiative) kernel application called Sweep3D. This application illustrates the level of accuracy that can be obtained, that cross-platform comparisons can be easily undertaken, and that the process benefits from a rapid evaluation time [4]. While extremely well-suited for managing locally distributed multi-computers, PACE functions map less well onto a wide-area environment, where the process of resource management becomes far more complex. In order to meet the requirements of performance prediction for grid computing environments, both scalability and adaptability of prediction systems 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 methodology has been developed for the management of large-scale distributed systems with highly dynamic behaviours [6]. The system consists of a hierarchy of homogenous agents where each agent can be considered both a service provider and a service requestor. Multiple homogeneous agents are organised into federated groups, which have capabilities of service advertisement and discovery. Different optimisation strategies and performance metrics are defined and used to improve the agent behaviours. Issues related to the utilisation of an agent-based methodology for grid resource management is discussed in [5] and [7]. In this paper, an initial implementation of the agent-based grid resource management system is described. The agent acts as a local resource manager, using a user-oriented scheduling algorithm and coupled with PACE to provide predictive capabilities regarding local grid resources and the services they can provide. At a meta-level, agents cooperate to advertise and discover services. A case study and corresponding experimental results are included in this paper. The results demonstrate the effect of management and scheduling obtained through the cooperation of an agent system coupled with the PACE performance prediction tool.

There are several solutions that currently address issues of grid resource management and scheduling. These include Globus [11], Legion [12], NetSolve [10], Condor [24], Ninf [22] and Nimrod/G [3]. While many of these projects utilise query-based mechanisms for resource discovery and advertisement [20], this work adopts an agent-based approach. The agent-based system allows agents to control the query process and make resource discovery decisions based on their own internal logic rather than rely on a fixed function query engine. The resource management system described in this work can provide hard quality of service (QoS) support as defined in [20]. However, unlike Nimrod/G, in which grid resource estimation is performed through heuristics and historical information, the performance prediction capabilities in this research are provided through PACE. In the Condor system, scheduling aims to maximise the utilisation of grid resources (resource-oriented); another approach, favoured by Nimrod and the research in this paper, is to provide a system which meets user specified deadlines (user-oriented).

Several new grid projects utilise existing distributed computing technologies such as CORBA [26] and Jini [2]. CORBA is not designed for the development of high performance computing applications, however there has been research aimed at providing CORBA based tools in a variety of different contexts. For example, in the work described in [30] a CORBA Commodity Grid (CoG) Kit enables the development of advanced Grid applications which maintain state-of-the-art software engineering practices and also reuse existing grid infrastructure. However, such technologies only enable resource sharing within a single organisation [14]. In [15] a computational community that supports the federation of resources from different organisations is described; this system is designed and implemented in Java and Jini. JiPANG (A Jini-based Portal Augmenting Grids) [28] is a portal system and a toolkit that provides a uniform access interface layer to a variety of grid systems. This allows the development of Jini-based systems on top of the Java platform.

Agent technologies have been used for the development of distributed software systems for several years [16]. An agent-based approach provides a clear high-level abstraction and a more flexible system implementation [17]. Multi-agent systems have recently been introduced in grid development and resource management. This work [18] includes a model for distributed awareness and a framework for the dynamic assembly of agents for the monitoring of network resources. An “Agent Grid” is described in [25] that integrates services and resources for establishing multi-disciplinary Problem Solving Environments (PSEs). Specialised agents contain behavioural rules and can modify these rules based on their interaction with other agents and with the environment in which they operate. The agent-based methodology used in this research can also be used for the integration of multiple services and resources. This is done using hierarchy of homogenous agents, rather than by utilising a collection of specialised agents.

The agent-based system described in this work is investigated through quantitative performance analysis using modelling and simulation techniques. Other grid simulation models have been built: Simgrid [9] is a

simulation toolkit for the study of scheduling algorithms for distributed applications; GridSim [21] investigates effective resource allocation techniques based on a computational economy. Grid performance modelling and simulation is a valuable tool, especially as current grid computing research has a limited number of practical grid environments and research test-beds with which to work [8].

The paper is organised as follows: section 2 introduces the PACE toolkit and corresponding methodology for performance prediction; in section 3, the system architecture and agent resource management system are described; a case study and experimental results are included in section 4; the paper concludes in section 5.

2. Performance Prediction Using PACE

The PACE performance prediction capabilities are essential to the system implementation. In this section the PACE toolkit, validation of the prediction capabilities and grid-enabling extensions are introduced.

2.1 PACE Toolkit

The main components of the PACE toolkit are shown in Figure 1; they include the application tools, the resource tools and an evaluation engine [23].

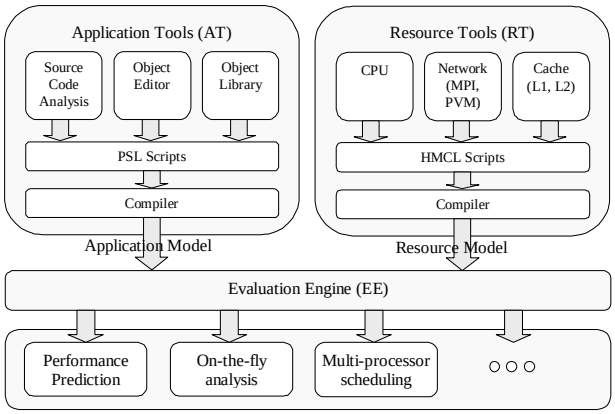

Figure 1. The PACE Toolkit

- Application Tools: A core component of this part of the toolkit is the performance specification language (PSL) which describes the performance aspects of an application and also its parallelisation. The Source Code Analyser is used to convert sequential source code components into performance descriptions. Users edit these descriptions using the object editor; existing objects can be retrieved from an Object Library. The performance descriptions are collated into application PSL scripts which are then compiled into an Application Model. This forms one of the inputs into the Evaluation Engine, which itself acts as a repository and analysis tool for the application-level performance information.
- Resource Tools: A hardware modelling and configuration language (HMCL) is used to define the computing environment in terms of its constituent performance model components. The Resource Tools provide several benchmarking programs to measure the performance of CPU, network, and memory aspects of hardware platforms respectively. The measurements are represented in HMCL scripts and combined to form a resulting Resource Model. This system-level performance information provides a second input to the Evaluation Engine.
- Evaluation Engine: The evaluation engine is the kernel of the PACE toolkit. The Evaluation Engine executes completed performance models to produce evaluation results, these include time estimates and trace information relating to the expected application behaviour.

Examples of the use of PACE include on-the-fly performance analysis for application execution steering [1], and dynamic multi-processor scheduling for efficient resource management [19].

2.2 Performance Validation

The performance prediction capabilities of PACE have been successfully demonstrated using the ASCI kernel application Sweep3D [4]. The validation experiments are carried out on two high performance platforms: an SGI Origin 2000 multiprocessor and a cluster of Sun Ultra1 workstations. The validation results show that:

- a good level of predictive accuracy can be achieved (the maximum predictive error is 20%, the average is approximately 10%);
- performance evaluation is rapid (typically seconds of CPU use) for a given system and problem size;
- from the results it is easy to obtain performance comparisons across different computational systems.

It has been shown that the PACE system can produce reliable performance information which can then be used in the investigation of application and system performance. In [19] it is shown that performance data produced by PACE can be used for the management of parallel and distributed systems. The PACE toolkit was

not however developed in the context of grid computing. In order to apply PACE to grid problems a number of modifications have been made. One of these modifications is the inclusion of transaction-based performance modelling.

2.3 Transaction-based Modelling

PACE operates by characterising an application in terms of its principle operations and its parallelisation strategy. It then couples these requirements with the hardware resources to obtain predictions of execution times. At present, costs are associated with individual machine level instructions which allows the toolkit to model detailed subtleties in an application code. This is considered too fine grained for grid applications and so variant of PACE is currently under development which uses 'transactions' as base units of work [27].

Figure 2 shows the original PACE structure (depicted on the left) which describes applications by means of a layered modelling language. While maintaining the same structure, the coarse-grained PACE (on the right) utilises transactions and transaction maps to characterise applications more rapidly. An application is represented as a number of transactions each of which encapsulate key components of an application code; a transaction map describes the interrelationships between these transactions.

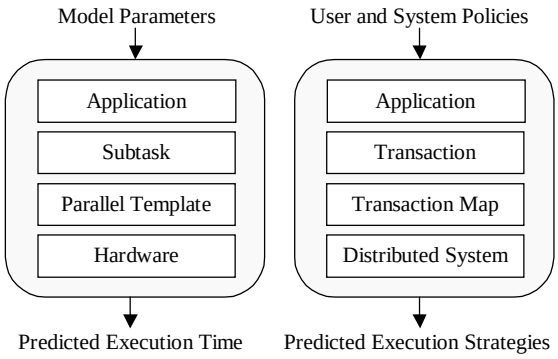

Figure 2. Coarse-grained PACE

Transaction-based application modelling enables efficient remote performance evaluation and prediction to be performed. This makes the toolkit highly appropriate for use in dynamic grid-like environments which consist of a number of heterogeneous systems. The aim of this research is to enable this approach without sacrificing performance accuracy. The kernel of the system is defined in a transaction definition language (TDL); this is described in detail in [29].

3. Agent-based Resource Management

An agent-based approach is used to integrate PACE functionality with grid resource management. In this section, both the overall architecture of the system and the structure of an individual agent are described.

3.1 System Architecture

An overview of the agent-based resource management architecture is illustrated in Figure 3. The main components include grid users, grid resources, agents and a performance monitor and advisor (PMA). These are introduced in detail below.

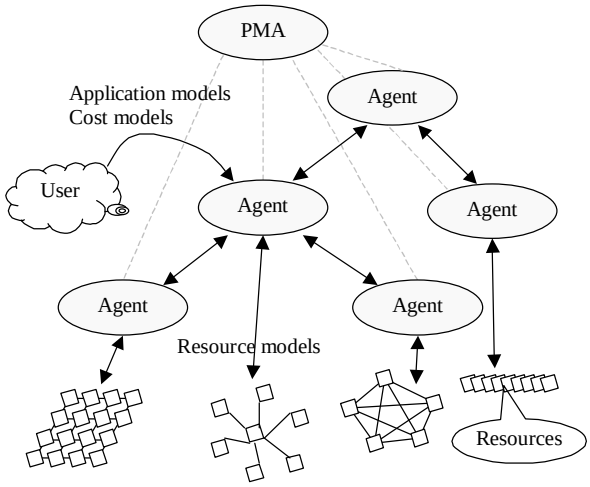

Figure 3. System Architecture

3.1.1. Grid Users

There are different kinds of users of a grid computing environment. These include grid service and tool developers, application developers and grid end users.

The developers of the tools, compilers, libraries, and so on implement the programming models and services used by application developers. MPI and PVM are included in these categories. Grid service and tool developers are a small group of grid users and are therefore not the main focus in the context of this work.

Application developers comprise those who construct grid-enabled applications using grid tools. There are different kinds of grid applications: distributed supercomputing, high throughput, on demand, data intensive and collaborative applications. The applications described in this work mainly refer to scientific

supercomputing applications; that is, large problems requiring a large amount of CPU and memory, etc., and which are (for the most part) written using MPI and PVM.

Most grid users, like most users of computers or networks today, will not write programs. Instead, these end users will use grid-enabled applications that make use of grid resources and services. In some cases, application developers will also be the end users of the applications that they develop. The grid users in Figure 3, and mentioned in the following sections, are considered to be scientists, who develop scientific supercomputing applications and who use them to solve large problems in a grid context.

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 can provide high performance computing capabilities for grid users. A resource can include Massive Parallel Processors (MPP), or a cluster of workstation or PCs. A grid resource can be considered a service provider of high performance computing capabilities.

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. Agents

Agents comprise the main components in the system. Each agent is viewed as a representative of a grid resource at a meta-level of resource management. An agent can therefore be considered a service provider of high performance computing capabilities. Agents are organised into a hierarchy. The hierarchy of homogenous agents provides a meta-level view of the grid resources. The service information of each grid resource can be advertised in the agent hierarchy (both upwards and downwards); agents can also cooperate with each other to discover available resources.

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.

An agent can choose to maintain different types of ACTs according to different sources of service information. For example, T_ACT is used to record the service information of local resources. Each agent can also have one L_ACT to record the service information received from its lower agents and one G_ACT from the upper agent. 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 simple protocol of service advertisement and discovery described above allows agents to be configured with different strategies, leading to different agent behaviours. Agents advertise and discover services according to their own internal logic; this allows a flexible implementation of grid resource management.

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 bridges the gap between grid application users and grid resources. A introduction to the use of agent-based service discovery for grid resource management can be found in [7]. The agent hierarchy

also allows scalability to be addressed. Service advertisement and discovery are processed stepwise between neighbouring agents only. This feature plays an important part in system scalability. Another important factor is the capacity for agents to be able to adjust their service advertisement and discovery behaviours, thus adapting to the highly dynamic grid environment. This is achieved through the introduction of a performance monitor and advisor.

3.1.4. Performance Evaluation

Performance issues arise from the dynamic nature of the grid resources. Unlike other work that focus on data representation and communication protocols, this research enables the performance of the agent system to be investigated quantitatively. Figure 3 shows the monitoring role of the PMA. The PMA observes agent communication traffic with the intention of improving agent performance.

Unlike facilitators or brokers in classical agent-based systems, the PMA is not central to the rest of the agents. 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.

The PMA is composed of a model composer and a simulation engine. 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.

The metrics used to describe the performance of the agent system include the service discovery speed, the overall system efficiency, the load balancing and also the discovery success rate. Corresponding performance optimisation strategies include the use made of the ACTs, the limits placed on the service lifetime and the scope of advertisement and discovery, agent mobility and the service distribution. These features are not described in detail in this paper; for more information of these and other aspects of the system, see [5].

3.2. Agent Structure

The structure of an agent - shown in Figure 4 - is divided into three component layers corresponding to communication, coordination and local management.

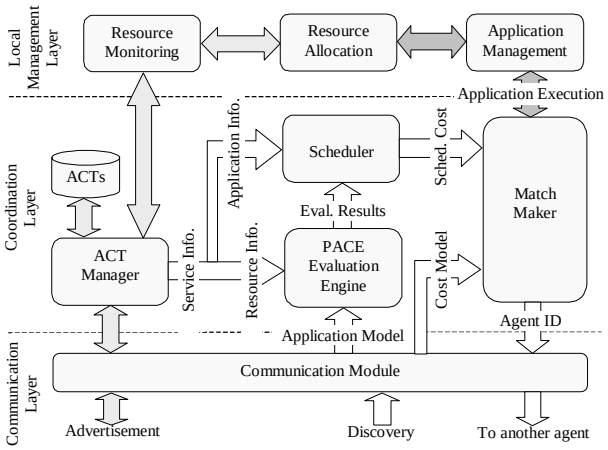

Figure 4. Agent Structure

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. They 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 allocation 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 the 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 resource 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 5 illustrates the content of this service information.

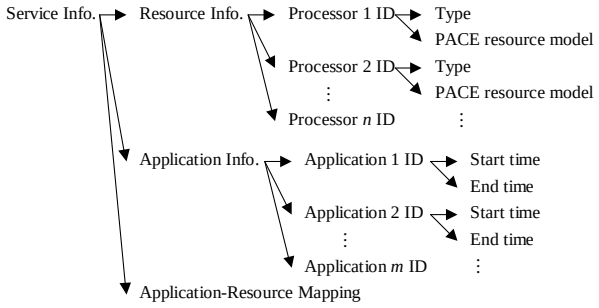

Figure 5. Service Information

Service information for a grid resource should include details of those aspects that have an impact on the performance of the resource and can therefore be used to evaluate its performance. Service information is therefore composed of resource information, application information, and the mapping between the applications and the resources.

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 this processor:

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

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

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

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

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

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

$$M = \{M_{ij} | i = 1,2,.....,n;\ j = 1,2,.....,m\}$$
$$M_{ij} = \begin{cases} 1 & if \quad P_i \in MA_j \\ 0 & if \quad P_i \notin MA_j \end{cases}$$

3.2.2. PACE Evaluation Engine

The request information consists of the PACE application model (*am*), which includes all performance related information of an application A_r. The requirements of the application are specified in a vector; this includes a number of metrics including the deadline for the execution of the application, t_{req}.

The PACE evaluation engine can produce performance prediction information based on the application model *am* and the resource information from the ACT manager, *ty*. Prediction data such as application execution time, *exet*, can be derived for the applications execution on the given resource.

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

Rather than running the application on all processors for a given grid resource P, an agent can select an appropriate subset of processors $\overline{P}$ (note that $\overline{P}$ cannot be an empty set Φ), this is evaluated and expressed as follows:

$$\forall \overline{P} \subseteq P, \overline{P} \neq \Phi, \overline{ty} \subseteq ty, \overline{ty} \neq \Phi, \overline{exet} = eval\left(\overline{ty}, am\right)$$

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. However, an agent can still schedule the required application execution based on this resource information. The function of the scheduler is to find the earliest time at which an application terminates on the resource 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 on a grid resource. The scheduler should consider all these possibilities and choose the earliest end time of the application execution. 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:

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

The earliest possible start time for application A_r on a selection of processors is the latest free time of all the selected processors if there are still applications running on the selected processors. If there is no application currently running on the selected processors, application A_r can be executed on these processors immediately. These is expressed as follows:

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

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

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

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

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

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

The scheduling algorithm described above is used in an initial system implementation. The complexity of the algorithm is determined by the number of possible processor selections, which can be calculated as:

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

It is clear that if the number of processors of a grid resource increases, the complexity of the local resource scheduling algorithm will increase exponentially. Though a local resource in grid environment can only have limited number of processors, this algorithm cannot scale well when the number of processors increases. Another factor is that the scheduling policy of this algorithm is to meet requirements from the user, instead of maximising the resource utilisation. There is no rescheduling process for previously scheduled applications. New algorithms need to be developed for such cases.

The importance of the efficiency of the PACE evaluation engine is clear. 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 in the use of PACE for grid resource management and in the provision of predictive QoS support for service discovery.

3.2.4. Matchmaker

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

In terms of application execution time, if $t_{req} \geq t_{sched}$, the corresponding resource can meet the user requirement. If the corresponding ACT item is in the T_ACT, a local resource is available for application

execution and the application execution command will be sent to the local management in the agent. Otherwise, the agent ID of the corresponding ACT item is returned, and the agent will dispatch the request to that agent via the agent ID.

If $t_{req} < t_{sched}$, the corresponding resource cannot meet the requirement from the user. The agent continues to look up other items in the ACTs until the available service information is found. The agent can look up different ACTs in turn and in the case of there being no available service information in the ACTs, the agent may submit or dispatch the request to its upper or lower agents for further discovery.

There may be many other metrics in the user supplied cost model and in this case the corresponding evaluation mechanisms should also be provided in each agent. Their implementation has parallels with the application execution time-based scheme and as a result is not discussed in further detail in this paper.

4. A Case Study

Experiments have been designed using the initial system implementation. There are two main parts in the design of the experiments. The system itself includes agents, resources and agent behaviour strategies used in the experiment. The automatic users of the system are also designed to send application execution requests with different frequencies, which add different workloads onto the system. Experimental results are also included to illustrate how the agent-based resource management system schedules applications onto available resources.

4.1. System Design

There are eight agents in the experimental system. The agent hierarchy is shown in Figure 6.

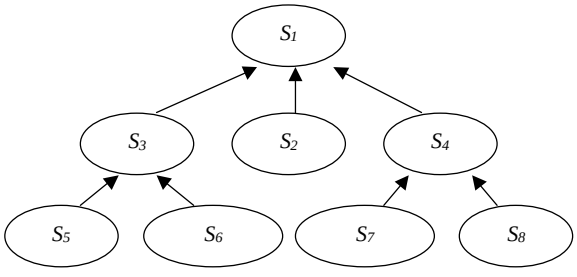

Figure 6. Case Study: Agent Hierarchy

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

Agents represent heterogeneous hardware resources containing sixteen processors per resource. As shown in Table 1, the resources range in their computational capabilities. The SGI multi-processor is the most powerful, followed by the Sun Ultra 10, 5, 1, and SparcStation in turn.

Table 1. Case Study: Resources

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

In the experimental system, each agent maintains a set of capability tables - T_ACT, L_ACT and G_ACT. T_ACTs are maintained by the event-driven data-push service advertisement. L_ACTs are updated every ten seconds using a data-pull. G_ACTs are also updated by data-pull, at a frequency of every thirty seconds. All of the agents employ identical strategies with the exception of the agent at the head of the hierarchy (S_1) that does not maintain a G_ACT.

4.2. Automatic User

The applications used in the experiments include typical scientific computing programs. Each application has been modelled and evaluated using PACE. An example of PACE predications for the system S_1, which represents the most powerful resource in the experiment, can be found in Figure 7. The predictions for the other systems follow a similar trend and are therefore omitted.

As shown in the figure, the run time of *sweep3d* decreases when the number of processors increases. At the same time the parallel efficiency also decreases. In fact, when the number of processors is more than 16, the run time does not improve any further. The results of the application *improc* show a different trend. Run time of *improc* decreases to an optimum of 8 processes – after which the run time then increases. Different applications have very different performance scenarios which has an important impact on the application scheduling results.

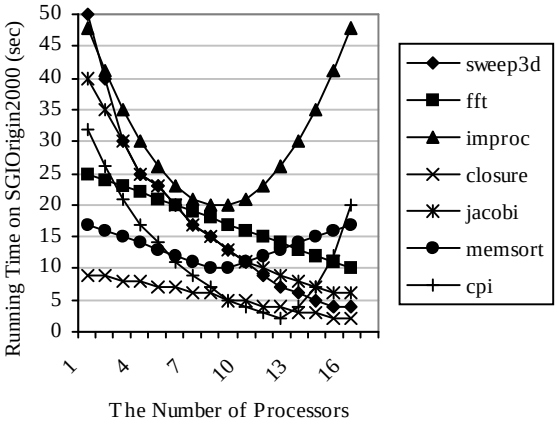

Figure 7. Case Study: Applications

An application execution request for one of the seven test applications is sent at random to an agent. Additionally, the required execution time for the application is also selected randomly from a given domain; the bounds of the application requirements can be found in Table 2.

Table 2. Case Study: Requirements

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

The automatic users are configured so that they send requests to the agents with different frequencies. As shown in Table 3, four experiments are designed with different system workloads.

Table 3. Case Study: Workloads

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

The interval of requests sent in each experiment is chosen randomly from a given domain, resulting in different average frequencies. Experiment No. 2, for example, lasts approximately 7 minutes; during this period, 149 requests are sent, that is one request sent every 3 seconds on average. The experimental results are discussed below.

4.3. Experiment Results

The experimental results can be found in Tables 4 and 5. These illustrate the number of applications accepted by a particular agent in each experiment and also the number of service discovery steps required to meet the overall workload. Tasks can be rejected by the system (failed) if insufficient resources are available to meet the requirements of the user.

Table 4. Experiment Results: Application Execution

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 5. Experiment Results: Service Discovery

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

Experiment No. 1

In this experiment, the system workload is light relative to the system capabilities, with an application request being sent every four seconds. This results in a balanced application distribution on the agents and the amount of requests that end unsuccessfully is small. Table 5 illustrates that 97% of service requests are completed with no service discovery.

Experiment No. 2

When the system workload becomes heavier, S_7 and S_8 cannot meet the computational requirements, and therefore submit the request to their upper agent S_4. This leads to a 6% increase in workload on S_4, and a 13% increase in the number of 1-step discovery requests. While heavily loaded, S_1 is sufficiently powerful to serve 19% of the total application requests.

Experiment No. 3

The system workload increases further. The 5% decrease of application executions on S_4 indicates that the local resource of S_4 has reached its capability. Requests submitted from S_7 and S_8 are passed to S_1, which leads to an 8% increase in the number of 2-step discovery processes. Service discovery amongst the agents becomes more active when the system workload increases.

Experiment No. 4

Experiment four represents a heavily loaded system. A decrease of 6% in accepted applications on S_1 indicates that S_1 has also reached its capacity. This doubles the number of failed requests. The number of 1-step discovery processes decreases by 8%, while 2-step and 3-step service discovery processes increase by 3%. This indicates that the whole system has reached its capability limit, resulting in more complex service discovery in order to find available resources.

With the workload increasing, the trends for the distributions of application execution and service discovery are shown in Figures 8 and 9 respectively. Some generalised information can be concluded.

Figure 8 illustrates that when the system workload increases, resources will reach their computational capabilities in turn. The more powerful a resource is, the later it reaches its limitation. The peaks appearing at the curves S_1 and S_4 indicate the time the corresponding resource reaches its limitation.

Another interesting phenomenon is that the system workload is balanced when it is extremely light in experiment 1 and heavy in experiment 4. In experiment 1, this occurs as a result of averagely sent requests and the need for no service discovery, while in experiment 4, the distribution of applications over the eight systems is balanced according to the system capabilities. The agent load corresponds to the computing capabilities of the respective resource. The agents S_1 and S_2, which represent the most powerful resources in the experimental environment, serve a larger percentage of the applications, this is followed by S_3, S_4, S_6 and S_5. Only a small percentage of the requests are serviced by the agents S_7 and S_8.

Figure 9 illustrates the trend in service discovery. As the system becomes more heavily loaded, the number of 0-step discoveries decreases with a related increase in the number of 1-step processes. Similarly, as load increases further the number of 1-step processes decreases with a rise in 2-step discovery. In general, when the workload increases, more complex service discovery processes occur while simpler ones disappear. However, because the number of agents in the experimental system is small, no more complex (more than 4-step) service discovery processes occur.

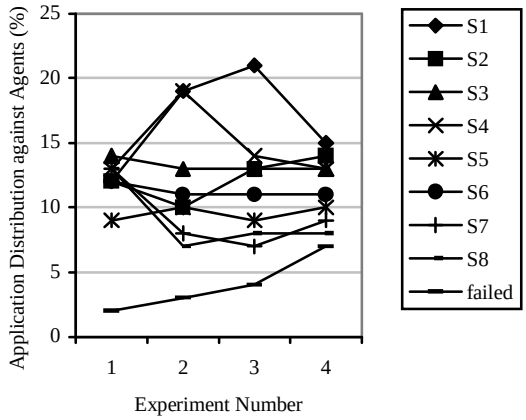

Figure 8. Experiment Results: Trend I

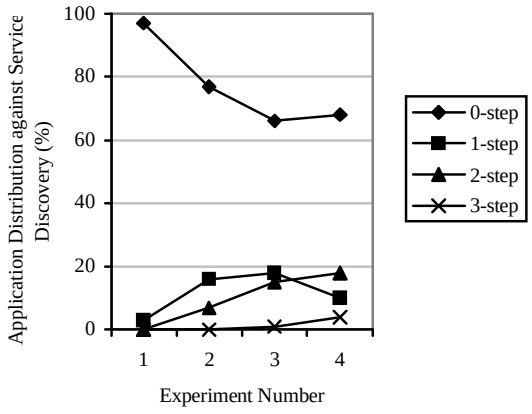

Figure 9. Experiment Results: Trend II

While this experimental system is far from a representative grid-sized environment, the experimental results demonstrate that the performance-driven agent-based resource management, utilising service advertisement and discovery, is effective for scheduling applications that require grid-like distributed resources.

5. Conclusions

The use of performance prediction techniques for agent-based resource management in grid environments is presented in this work. An initial implementation of an agent-based resource management system is described. A case study is described in detail to demonstrate the efficiency of the resource management and scheduling capabilities of the system. The main features in this work include:

- hard QoS support using the PACE performance prediction capabilities;
- agent-based dynamic resource advertisement and discovery;
- simulation-based quantitative grid performance analysis;
- and user-oriented scheduling of local grid resources.

Future work will focus on the system enhancement. Some existing standards, languages, tools and protocols can be utilised. For example, the agents and the PMA can be developed using Java and an XML format for data representation. An agent communication language (ACL) can be used to allow agents to communicate with each other at a higher-abstracted knowledge level. The system will also be integrated with current grid standard toolkit Globus.

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] A. M. Alkindi, D. J. Kerbyson, G. R. Nudd, and E. Papaefstathiou, "Optimisation of Application Execution on Dynamic Systems", Future Generation Computer Systems, Vol. 17, No. 8, pp. 941-949, 2001.

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

[3] R. Buyya, D. Abramson, and J. Giddy, "Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid", in Proc. 4th Int. Conf. on High Performance Computing in Asia-Pacific Region, Beijing, China, 2000.

[4] J. Cao, D. J. Kerbyson, E. Papaefstathiou, and G. R. Nudd, "Performance Modelling of Parallel and Distributed Computing Using PACE", in Proc. 19th IEEE Int. Performance, Computing and Communication Conf., Phoenix, Arizona, USA, pp. 485-492, 2000.

[5] J. Cao, D. J. Kerbyson, and G. R. Nudd, "Performance Evaluation of an Agent-Based Resource Management Infrastructure for Grid Computing", in Proc. 1st IEEE Int. Symp. on Cluster Computing and the Grid, Brisbane, Australia, pp. 311-318, 2001.

[6] J. Cao, D. J. Kerbyson, and G. R. Nudd, "High Performance Service Discovery in Large-scale Multi-agent and Mobile-agent Systems", Int. J. Software Engineering and Knowledge Engineering, Special Issue on Multi-Agent Systems and Mobile Agents, World Scientific Publishing, Vol. 11, No. 5, pp. 621-641, 2001.

[7] J. Cao, D. J. Kerbyson, and G. R. Nudd, "Use of Agent-based Service Discovery for Resource Management in Metacomputing Environment", in Proc. 7th Int. Euro-Par Conf., LNCS 2150, Springer-Verlag, pp. 882-886, 2001.

[8] J. Cao, "Agent-based Resource Management for Grid Computing", PhD Dissertation, Dept. of Computer Science, Univ. of Warwick, 2001.

[9] H. Casanova, "Simgrid: A Toolkit for the Simulation of Application Scheduling", in Proc. 1st IEEE Int. Symp. on Cluster Computing and the Grid, Brisbane, Australia, pp. 430-437, 2001.

[10] H. Casanova, and J. Dongarra, "Applying NetSolve's Network-Enabled Server", IEEE Computational Science & Engineering, Vol. 5, No. 3, pp. 57-67, 1998.

[11] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke, "A Resource Management Architecture for Metacomputing Systems", in Proc. IPPS/SPDP '98 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, Vol. 15, No. 5, pp. 583-594, 1999.

[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 Proc. of 7th Int. Euro-Par Conf., LNCS 2150, Springer-Verlag, pp. 855-863, 2001.

[16] N. R. Jennings, and M. J. Wooldridge (eds), Agent Technology: Foundations, Applications, and Markets, Springer-Verlag, 1998.

[17] N. R. Jennings, "An Agent-based Approach for Building Complex Software Systems", Communications of the ACM, Vol. 44, No. 4, pp. 35-41, 2001.

[18] K. Jun, L. Boloni, K. Palacz, and D. C. Marinescu, "Agent-Based Resource Discovery", in Proc. 9th IEEE Heterogeneous Computing Workshop, 2000.

[19] 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 Proc. 6th Int. Euro-Par Conf., LNCS 1900, Springer-

Verlag, pp. 149-159, 2000.
[20] K. Krauter, R. Buyya, and M. Maheswaran, "A Taxonomy and Survey of Grid Resource Management Systems", to appear in Software: Practice and Experience, 2001.
[21] M. Murshed, R. Buyya, and D. Abramson, "GridSim: A Toolkit for the Modelling and Simulation of Global Grids", Technical Report, Monash University, 2001.
[22] H. Nakada, H. Takagi, S. Matsuoka, U. Nagashima, M. Sato, and S. Sekiguchi, "Utilizing the Metaserver Architecture in the Ninf Global Computing System", in Proc. High-Performance Computing and Networking, LNCS 1401, Springer-Verlag, pp. 607-616, 1998.
[23] 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", Int. J. High Performance Computing Applications, Special Issues on Performance Modelling, Sage Science Press, Vol. 14, No. 3, pp. 228-251, 2000.
[24] R. Raman, M. Livny, and M. Solomon, "Matchmaking: Distributed Resource Management for High Throughput Computing", in Proc. 7th IEEE Int. Symp. on High Performance Distributed Computing, 1998.
[25] O. F. Rana, and D. W. Walker, "The Agent Grid: Agent-Based Resource Integration in PSEs", in Proc. 16th IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation, Lausanne, Switzerland, 2000.
[26] D. Slama, J. Garbis, and P. Russell, Enterprise Corba, Prentice Hall, 1999.
[27] D. P. Spooner, J. D. Turner, J. Cao, S. A. Jarvis, and G. R. Nudd, "Application Characterisation Using a Lightweight Transaction Model", in Proc. 17th Annual UK Performance Engineering Workshop, Leeds, UK, pp. 215-225, 2001.
[28] T. Suzumura, S. Matsuoka, and H. Nakada, "A Jini-based Computing Portal System", in Proc. SuperComputing 2001.
[29] 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, January 2002.
[30] S. Verma, M. Parashar, J. Gawor, and G. von Laszewski, "Design and Implementation of a CORBA Commodity Grid Kit", in Proc 2nd IEEE Workshop on Grid Computing, 2001.

Biographies

Junwei Cao is currently a Research Assistant in the Department of Computer Science at the University of Warwick, UK. His research interests include resource management for grid computing, performance evaluation of parallel and distributed computing, multi-agent systems and object-oriented system analysis and design. He is now working on the project "Use of Performance Prediction Techniques for Grid Management", sponsored by a grant from the NASA AMES Research Centre. He received his PhD on Computer Science in 2001 from Warwick and his thesis focuses on agent-based resource management for grid computing. Before joining Warwick in 1998, he was a research student in Tsinghua University, P. R. China, and took part in the system analysis, design and implementation of several projects, using structural, object-oriented and agent-based methods respectively. He received his BEng and MSc in 1996 and 1998 from Tsinghua. He is a member of the IEEE Computer Society and the ACM.

Stephen A. Jarvis is a Lecturer in the Department of Computer Science and member of the High Performance Systems Group. He has over 20 publications (including one book) in the area of software and performance evaluation. Previously at the Oxford University Computing Laboratory, he worked on performance evaluation tools for a number of programming paradigms including the Bulk Synchronous Parallel (BSP) programming library – with Oxford Parallel and Sychron Ltd. - and the Glasgow Haskell Compiler – with Glasgow University, Durham University and Microsoft Research, Cambridge. He is now working with IBM Hursley (UK) and IBM Watson (US) on performance aspects of GRID computing.

Daniel P. Spooner is currently a PhD student in the Department of Computer Science at the University of Warwick, UK and is a member of the High Performance Systems Group. His primary research interests are in the application of performance prediction techniques to optimize wide-area resource scheduling in Grid environments. Other interests include distributed network management architectures and high performance web services.

James D. Turner, is a final year PhD student in Computer Science in the High Performance Systems Group, University of Warwick, UK. His main research areas include high performance Java and the performance prediction and performance measurement of distributed applications. This work is being used in a scalable scheduling environment for GRID architectures, an agent model for the discovery and advertisement of distributed resources, and an environment for the efficient routing of web service requests. His work has contributed to the development of the ARM standard for predicting the performance of applications, and using such information to provide efficient resource allocation dependant upon a number of QoS constraints.

Darren J. Kerbyson is with Los Alamos National Laboratory. He is actively involved in modelling the performance of tera-scale systems in the ASCI programme. Before joining Los Alamos, he was a senior member of staff in the Department of Computer Science at the University of Warwick. His areas of interest include techniques for performance prediction, large-scale parallel and distributed processing systems, as well as image analysis. He has published over 50 papers in these areas over the last 10 years. Dr. Kerbyson is a member of the ACM and the IEEE Computer Society.

Graham R. Nudd is Head of the High Performance Systems Group and Chairman of the Computer Science Department at the University of Warwick, UK. 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, CA.