In Proceedings of 17th IEEE International Parallel & Distributed Processing Symposium (IPDPS 2003), Nice, France, April 2003, 10 pages.

Agent-Based Grid Load Balancing Using Performance-Driven Task Scheduling

Junwei Cao[*1], Daniel P. Spooner[†], Stephen A. Jarvis[†], Subhash Saini[‡] and Graham R. Nudd[†]
[*]*C&C Research Laboratories, NEC Europe Ltd., Sankt Augustin, Germany*
[†]*Department of Computer Science, University of Warwick, Coventry, UK*
[‡]*NASA Ames Research Center, Moffett Field, California, USA*
Email address for correspondence: cao@ccrl-nece.de

Abstract

Load balancing is a key concern when developing parallel and distributed computing applications. The emergence of computational grids extends this problem, where issues of cross-domain and large-scale scheduling must also be considered. In this work an agent-based grid management infrastructure is coupled with a performance-driven task scheduler that has been developed for local grid load balancing. Each grid scheduler utilises predictive application performance data and an iterative heuristic algorithm to engineer local load balancing across multiple processing nodes. At a higher level, a hierarchy of homogeneous agents are used to represent multiple grid resources. Agents cooperate with each other to balance workload in the global grid environment using service advertisement and discovery mechanisms. A case study is included with corresponding experimental results to demonstrate that both local schedulers and agents contribute to overall grid load balancing, which significantly improves grid application execution performance and resource utilisation.

1. Introduction

A computational grid is an emerging computing infrastructure that enables effective access to high performance computing resources [11]. Resource management and scheduling are key grid services, where issues of load balancing represent a common concern for most grid infrastructure developers. While these are established research areas in parallel and distributed computing, grid computing environments present a number of new challenges, including:

- Cross-domain: The process of grid resource scheduling encompasses multiple administrative domains, where one domain may be unaware of the resources offered by another. The lack of central ownership and control means that it is difficult to map an entire set of tasks to permutations of the available resources.
- Large-scale: As a grid can encompass a large number of high performance computing resources that are not entirely dedicated to the environment, computational capabilities can vary significantly over time. These dynamic properties must be addressed when implementing grid information and resource management infrastructures.

In our previous work an agent-based methodology was developed for large-scale distributed software systems with highly dynamic behaviours [6, 7]. This has been used in the implementation of an agent-based resource management infrastructure for grid computing [8, 9], where each agent represents a local grid resource and acts as a service provider of high performance computing power. Agents cooperate with each other to discover available resources for tasks using a technique of service advertisement and discovery.

The research presented in this paper adopts the agent-based methodology to grid load balancing, achieved by coupling the agent system with a performance-driven task scheduler that has been developed for local grid load balancing. Each local scheduler uses an iterative heuristic algorithm based on the predictive performance data for each application. The algorithm aims to minimise makespan and processor idle time, whilst meeting the deadlines set 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 or processors available in the local domain. At a higher level, the hierarchy of homogeneous agents are responsible for dispatching tasks from grid users to each of the available local grid schedulers; this is orchestrated

[1] This work was carried out when the author was with the University of Warwick.

by some matchmaking and decision-making policies that are also driven by performance prediction data. Each agent is only aware of neighbouring agents and service advertisement and discovery requests are only processed among neighbouring agents, which provides the possibility for scaling over large wide-area grid architectures.

Application performance prediction provides the essential functionality that enables the grid load balancing capabilities described in this work. The PACE toolkit [15] is used to provide this capability for both the local schedulers and the grid agents. The main components of the PACE toolkit include application tools, resource tools, and an evaluation engine. The PACE evaluation engine can combine application and resource models at run time to produce performance data (such as total execution time). In the work described in [5] an ASCI (Accelerated Strategic Computing Initiative) kernel application, Sweep3D, is used to illustrate the performance prediction capabilities of PACE. The validation results show that a high level of accuracy can be obtained, cross-platform comparisons can be easily undertaken, and the process benefits from a rapid evaluation time. These features allow PACE predictive data to be used *on-the-fly* for grid resource load balancing.

There are a number of approaches to scheduling and load balancing parallel and distributed systems. Unlike batch queuing systems, such as Condor [13] and LSF [19], that address resource management within a local grid, the local grid scheduler described in this work is based on application performance prediction. While AppLeS [4] and Ninf [14] are also based on performance evaluation techniques, they utilise the NWS [18] resource monitoring service, as opposed to the performance prediction capabilities provide by the PACE toolkit. Nimrod [3] has a number of similarities to this work, including a parametric engine and heuristic algorithms [1] for scheduling jobs. The kernel of our local grid scheduler is a genetic algorithm (GA) [20]. Some existing systems use the Globus toolkit [10] to integrate with the grid computing environment, including Condor-G [12], Nimrod/G [2] and Ninf-G [17]. The scheduler introduced in this work is distributed for grid computing using an agent-based methodology; where 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 resource discovery is also used in [16], where each agent either represents a user application, a resource, or a matchmaking service. Rather than using a collection of predefined specialised agents, this work uses a hierarchy of homogeneous agents that can be reconfigured with different roles at run time.

The paper is organised as follows: section 2 introduces the theory and implementation of the schedulers for local load balancing; in section 3, the agent-based implementation of a grid load balancing system is described; a case study with experimental results is included in section 4 and the paper concludes in section 5.

2. Performance-Driven Task Scheduling

In this work, a local grid is considered to be a network of processing nodes (such as a multiprocessor or a cluster of workstations). Load balancing issues are addressed at this level using task scheduling algorithms driven by PACE performance predictions.

2.1 Genetic Algorithm

Consider a grid resource P with n processing nodes. A PACE resource model ρ_i can be used to describe the performance information of each processor P_i.

$$P = \{P_i \mid i = 1,2,.....,n\} \quad (1)$$

$$\rho = \{\rho_i \mid i = 1,2,.....,n\} \quad (2)$$

A set of parallel tasks T is considered to be run on P, where a PACE application model σ_j includes the performance related information of each task T_j. Additionally, T_j is specified with a requirement of the application execution deadline δ_j from the user.

$$T = \{T_j \mid j = 1,2,.....,m\} \quad (3)$$

$$\sigma = \{\sigma_j \mid j = 1,2,.....,m\} \quad (4)$$

$$\delta = \{\delta_j \mid j = 1,2,.....,m\} \quad (5)$$

A schedule is defined by a set of nodes $\overline{P_j} \subseteq P$ (corresponding $\overline{\rho_j} \subseteq \rho$) allocated to each task T_j and a start time τ_j at which the allocated nodes all begin to execute the task in unison. The execution time for each task T_j is a function, $t_x(\overline{\rho_j}, \sigma_j)$, provided by the PACE evaluation engine. The completion time η_j of each task T_j is defined as:

$$\eta_j = \tau_j + t_x(\overline{\rho_j}, \sigma_j). \quad (6)$$

The makespan, ω, for a particular schedule, which represents the latest completion time of any task, is subsequently defined as:

$$\omega = \max_{1 \le j \le m} \{\eta_j\}, \quad (7)$$

The goal is to minimise function (7) with respect to the schedule, at the same time $\forall j, \eta_j \leq \delta_j$ 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. 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, S_k, 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.

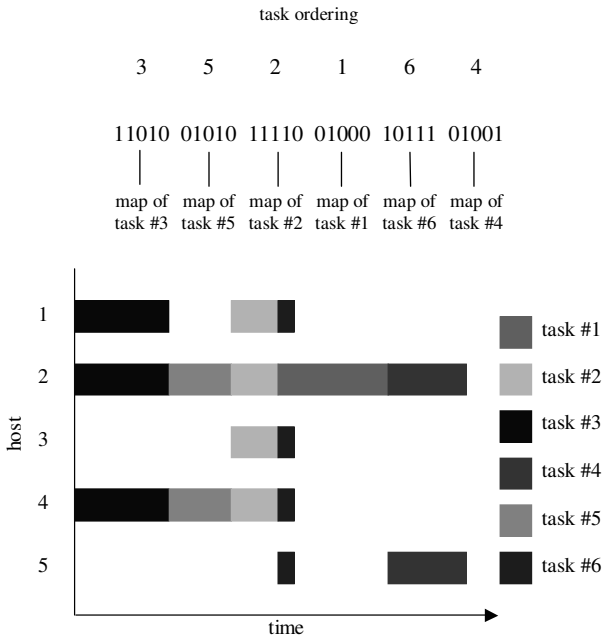

Figure 1. An Example Coding Scheme and Corresponding Gantt Chart

The coding scheme we have developed for this problem consists of two parts: an ordering part, which specifies the order in which the tasks are to be executed and a mapping part, which specifies the allocation of processing nodes to each task. The ordering of the task-allocation sections in the mapping part of the string is commensurate with the task order. An example of a solution string and its associated schedule are shown in Figure 1. The execution times of the various tasks are provided by the performance prediction system and are associated with the task object for evaluation by the fitness function f_v.

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 represented by any solution string S_k. The nature of the idle time should also be taken into account. Idle time at the front of the schedule is particularly undesirable as this is the processing time which will be wasted first, and is least likely to be recovered by further iterations of the GA or if more tasks are added. Solutions that have large idle times are penalised by weighting pockets of idle time to give φ_k, which penalises early idle time more than later idle time. The contract penalty θ_k is derived from the expected deadline times δ and task completion time η. The cost value for a schedule, represented by a solution string S_k, is derived from these metrics and their impact predetermined by:

$$f_c^k = \frac{W^m \omega_k + W^i \varphi_k + W^c \theta_k}{W^m + W^i + W^c} \tag{8}$$

The cost value is then normalised to a fitness value using a dynamic scaling technique:

$$f_v^k = \frac{f_c^{\max} - f_c^k}{f_c^{\max} - f_c^{\min}}, \tag{9}$$

where f_c^{max} and f_c^{min} represent the best and worst cost value in the scheduling set.

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.

2.2 System Implementation

A local grid scheduling system is developed in Java to implement the algorithm described above. It uses the PACE evaluation engine for application performance prediction data with several other modules included for

task management, execution, and resource monitoring. The system implementation is illustrated in Figure 2.

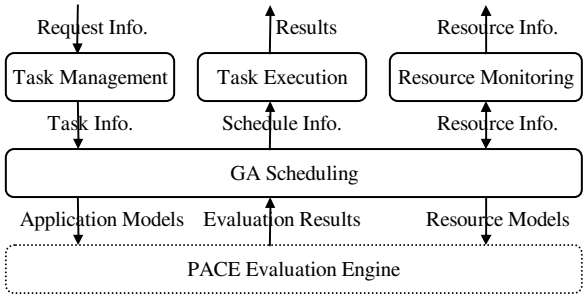

Figure 2. The System Implementation of a Performance-Driven Local Grid Scheduler

- *Task management*. Requests are passed to the task management module where they queue for scheduling and execution. Each task is given a unique identification number and awaits the attention of the GA scheduler. Task management also interfaces with the operations on the task queue, including adding, deleting or inserting tasks. The task queue is regarded by the GA scheduling as the optimisation set of tasks *T*.
- *Resource monitoring*. The resource monitoring is responsible for gathering statistics concerning the process nodes on which tasks may execute. These statistics include availability, load average and idle time. Currently, only host availability is supported, where the resource monitor queries each known node every five minutes. This is provided to the GA scheduler as the currently available resources *P* on which tasks can be scheduled. Resource monitoring is also responsible for organising the GA scheduling results and resource availabilities into service information that can be advertised. This is only used when a grid environment is considered and will be introduced in the following sections.
- *Task execution*. This is responsible for executing the program associated with a task on a scheduled list of processors. Currently, MPI and PVM based programs are supported and the executable programs must be pre-compiled and available in all local file systems. It is envisaged that this will be extended as the system matures.

The PACE evaluation engine is used to produce predictive data on the task execution time given appropriate application and resource models. While PACE evaluations complete relatively quickly (usually in the order of a few tenths of a second), as the search space of the GA is large, it is not usually possible to pre-calculate all required performance estimates ahead of time. A demand-driven evaluation scheme is used, coupled with a cache of past evaluations. When a particular evaluation result is requested, the cache is searched. If the result already exists, it is returned to the scheduler. Otherwise the PACE evaluation engine executes and the result is added to the cache before being returned.

The genetic algorithm and corresponding system implementation provide a fine-grained solution to dynamic load balancing on local grid resources and aim to maximise resource utilisation and meet task deadlines. However, this cannot provide load balancing services to a global grid since the GA does not scale to thousands of processing nodes and tasks. An additional mechanism for distributing over wide-area grid resources is required. This is achieved through the introduction of an agent-based resource advertisement and discovery framework.

3. Agent-based Grid Load Balancing

The problem which is considered in this section is the discovery of grid resources that provide the optimum execution performance for globally grid-submitted tasks. This research is not aimed at co-allocating a task onto more than one grid resource. Most MPI or PVM based programs use tightly coupled parallelism and it is difficult to achieve high performance when relatively slow communication mediums are involved.

3.1 Agent-based Service Discovery

While agent-based service discovery and its application to grid resource management has been described in detail in previous work [6, 7, 8, 9], a brief introduction is provided which demonstrates how the concepts apply to grid load balancing. This is extended to include a detailed system implementation in section 3.2.

A performance-driven scheduler (described in section 2) is responsible for managing the processing nodes at a local level and scheduling the incoming tasks to achieve local load balancing. Each agent provides a high-level representation of each local scheduler and therefore characterises these local resources as high-performance computing service providers in the wider grid environment. This higher-level representation is enhanced by organising the agents into a hierarchy, where 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.

- *Service advertisement*. An agent can advertise service information to both upper and lower agents.

Different strategies can be used to control these processes, which has an impact on the system efficiency. Service information can be *pushed* to or *pulled* from other agents, a process that is triggered by system events or through periodic updates.

- *Service discovery*. Discovering available services is also a cooperative activity. Within each agent, its own service is evaluated first. If the requirement can be met locally, the discovery ends successfully. Otherwise service information from both upper and lower agents 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 (representing a request for a computing resource which is not supported by the available grid).

While the processes 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 grid resource. While this may decrease the load balancing effect, the trade-off is reasonable as grid users prefer to find a satisfactory resource as fast and as local as possible.

The advertisement and discovery mechanisms also allow 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 might act as a potential bottleneck. While further work is necessary to test the scalability of the system, the experiments described in section 4 illustrate the contribution of both local schedulers and agents to the process of grid load balancing. These experiments use the initial implementation as described below.

3.2 System Implementation

The system architecture can be found in Figure 3. Agents are implemented using Java and data are represented in an XML format. Different PACE tools are integrated for different uses. The local schedulers are performance-driven - as described in section 2. The detail of how the agents utilise the PACE performance data and work with local schedulers to achieve grid load balancing is described.

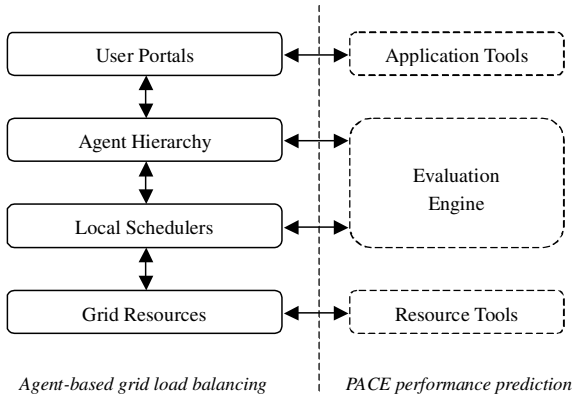

Figure 3. The System Architecture for Agent-Based Grid Load Balancing

Local schedulers are responsible for the submission of service information to the agents (see section 2.2). An example of this service information can be found in Figure 4.

```
<agentgrid type="service">
  <agent>
    <address>gem.dcs.warwick.ac.uk</address>
    <port>1000</port>
  </agent>
  <cluster>
    <address>gem.dcs.warwick.ac.uk</address>
    <port>10000</port>
    <type>SunUltra10</type>
    <nproc>16</nproc>
    <environment>mpi</environment>
    <environment>pvm</environment>
    <environment>test</environment>
    <freetime>Nov 15 04:43:10 2001</freetime>
  </cluster>
<agentgrid>
```

Figure 4. An Example Service Information

The identity of a local scheduler and its corresponding agent is provided by a tuple of the *address* and *port* used to initiate communication. The hardware model and the number of processors is also provided. The example specifies a single grid resource, in this case a cluster of 16 SunUltra10 workstations. To simplify the problem, the processors within each grid node are configured to be homogeneous. The application execution environments that are supported by the current implementation of the local schedulers 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 the predictive application execution times are scheduled and assumed to be accurate. The latest GA scheduling makespan ω indicates the earliest (approximate) time that corresponding processors become available for more tasks. Due to the effect of GA load balancing, it is

reasonable to assume that all processors within a grid 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 in Figure 5.

```
<agentgrid type="request">
  <application>
    <name>sweep3d</name>
    <binary>
      <file>binary/sweep3d</file>
      <inputfile>binary/input.50</inputfile>
    </binary>
    <performance>
      <datatype>pacemodel</datatype>
      <modelname>model/sweep3d</modelname>
    </performance>
  </application>
  <requirement>
    <environment>test</environment>
    <deadline>Nov 15 04:43:17 2001</deadline>
  </requirement>
  <email>junwei@dcs.warwick.ac.uk</email>
</agentgrid>
```

Figure 5. An Example Request Information

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 σ_r, which is generated using the application tools embedded in the user portal. 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 δ_r should be specified. Currently the user's email address is used as the contact information to which the results of the submission are posted.

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. This match is straightforward if a local grid scheduler can provide the required application execution environment. The expected execution completion time for a given task on a given resource can be estimated using:

$$\eta_r = \omega + \min_{\forall \bar{P} \subseteq P, \bar{P} \neq \Phi, \rho \subseteq \rho, \bar{\rho} \neq \Phi} \left\{ t_x(\bar{\rho}, \sigma_r) \right\}. \quad (10)$$

For a homogeneous local grid resource, the PACE evaluation function is called n times. If $\eta_r \leq \delta_r$, the resource is considered to be able to meet the required deadline. Otherwise, an appropriate local resource is not considered available for the incoming task. This performance estimation of local grid resources at the agent level is simple but efficient. However, when the task is dispatched to an available resource, the real situation may differ from the scenario considered in (10). The local GA scheduler may change the task order and advance or postpone a specific task execution in order to balance the workload on different processors, and in so doing maximise resource utilisation whilst 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 scheduler. 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.

3.3. Performance Metrics

There are a number of performance criteria that can be used to describe grid resource management and scheduling systems. What is considered as high performance depends on the system requirements. In this work three common statistics are investigated quantitatively and used to characterise the effect of grid load balancing. These include: average advance time of application execution completion ε, average resource utilisation rate υ and load balancing level β.

During a period of time t, a set of M parallel tasks T are scheduled onto grid resources P with N processing nodes. Note that the processing nodes may include those at all local grids. The final scheduling scenario can be described using the allocation to each task T_j (with deadline δ_j) a set of nodes $\overline{P_j} \subseteq P$ and a time domain $[\tau_j, \eta_j]$ during which the allocated nodes are simultaneously utilised for task execution.

The average advance time of application execution completion ε can be calculated directly using:

$$\varepsilon = \frac{\sum_{j=1}^{M} (\delta_j - \eta_j)}{M}, \quad (11)$$

which is negative when most deadlines fail. The resource utilisation rate υ_i of each processing node P_i is calculated as follows:

$$\upsilon_i = \frac{\sum_{\forall j, P_i \in \overline{P_j}} (\eta_j - \tau_j)}{t}. \quad (12)$$

The average resource utilisation rate υ of all processing nodes P is:

$$\upsilon = \frac{\sum_{i=1}^{N} \upsilon_i}{N}, \tag{13}$$

where υ is in the range 0 … 1. The mean square deviation of υ_i is defined as:

$$d = \sqrt{\frac{\sum_{i=1}^{N} (\upsilon - \upsilon_i)^2}{N}}, \tag{14}$$

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

$$\beta = 1 - \frac{d}{\upsilon}. \tag{15}$$

The most effective load balancing is achieved when d equals zero and β equals 1. The three aspects of the system described above are interrelated. For example, if the grid workload is balanced across all the processing nodes, the resource utilisation rate is usually high and the tasks finish quickly. These are illustrated in the case study described in the next section.

4. A Case Study

Three experiments have been designed using the system implementations described in sections 2 and 3 to evaluate different load balancing configurations. Experimental results regarding the three performance metrics are also included to illustrate the contribution of local schedulers and agents for grid load balancing to improve both resource utilisation and application execution.

4.1. Experimental Design

The experimental system is configured with twelve agents, illustrated by the hierarchy shown in Figure 6. These agents are named S_1……S_{12} (for the sake of brevity) and represent heterogeneous hardware resources containing sixteen processing nodes per resource. As shown in Figure 6, 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 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 every ten 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 have an upper agent.

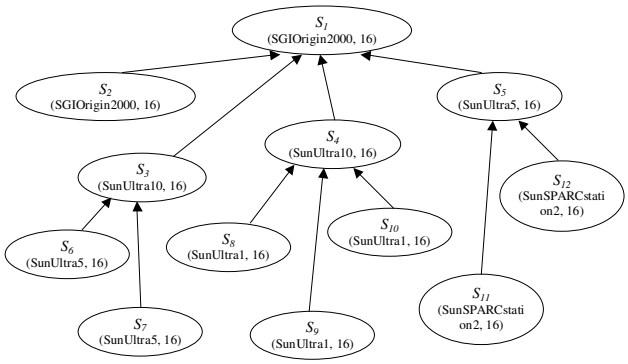

Figure 6. Case Study: Agents and Resources

The applications used in the experiments include typical scientific computing programs. Each application has been modelled and evaluated using PACE. An example of the PACE predications for the system S_1 (which represents the most powerful resource in the experiment) can be found in Table 1.

Table 1. Case Study: Applications and Requirements

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
sweep3d [4,200]	50	40	30	25	23	20	17	15	13	11	9	7	6	5	4	4
fft [10,100]	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
improc [20,192]	48	41	35	30	26	23	21	20	20	21	23	26	30	35	41	48
closure [2, 36]	9	9	8	8	7	7	6	6	5	5	4	4	3	3	2	2
jacobi [6,160]	40	35	30	25	23	20	17	15	13	11	10	9	8	7	6	6
memsort [10,68]	17	16	15	14	13	12	11	10	10	11	12	13	14	15	16	17
cpi [2,128]	32	26	21	17	14	11	9	7	5	4	3	2	4	7	12	20

As shown in the table, different applications have different performance scenarios which has a significant impact on the task scheduling results. During each experiment, requests for one of the seven test applications are sent at one second intervals to randomly selected agents. The required execution time deadline for the application is also selected randomly from a given domain; the bounds of the application requirements can also be found in Table 1. The request phase of each experiment lasts for ten minutes during which 600 task execution requests are sent out to the agents. While the selection of agents, applications and requirements are random, the seed is set to the same so that the workload for each experiment is identical.

While the experiments use the same resource configurations and application workloads, different combinations of local grid scheduling algorithms and global load balancing mechanisms are applied as shown in Table 2.

Table 2. Case Study: Experimental Design

	Experiment Number		
	1	2	3
FIFO algorithm	√		
Iterative heuristic algorithm		√	√
Agent-based service discovery			√

The GA algorithm and agent-based service discovery are introduced respectively in sections 2 and 3. The GA scheduling is compared with the results from a simpler first-in-first-out (FIFO) algorithm. The FIFO scheduling does not change the order of tasks. Each task is scheduled according to the time at which it arrives (also driven by the PACE predictive data). All of the possible resource allocations (a total of 2^{16}-1 possibilities) are tried. As soon as the current best solution is found, it is fixed and will not change as new tasks enter the system. The three performance metrics are used to measure the load balancing effect at both the local and global levels.

4.2. Experimental Results

The experimental results are given in Table 3; this includes the three metrics applied to each agent and to all the grid resources in the system.

Table 3. 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

Experiment No. 1

In the first experiment, the FIFO algorithm is used by each local grid scheduler with no supporting higher-level agent-based mechanism provided. The algorithm does not consider makespan, idletime or deadline. Each scheduler receives approximately 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. The overall average delay for task execution is approximately 8 minutes. It is apparent that the high performance platforms are not utilised effectively, and the lack of proper scheduling overloads resources like S_{11} that is only 44% utilised. The average utilisation of grid resources is only 26%. The workload for each of the processing nodes 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%.

Experiment No. 2

In experiment 2, the GA algorithm is used in place of the FIFO algorithm although no higher-level agent-based load-balancing mechanism is applied. The algorithm aims to minimise makespan and idletime, whilst meeting deadlines. Compared to those of experiment 1, almost all metrics are improved. Task executions are completed earlier and the average task execution delay is reduced to approximately 5 minutes. However, resources such as S_{11} and S_{12} remain overloaded and the GA is not able to find solutions that satisfy all the deadlines. Generally, resources are better utilised as a result of the GA scheduling, such as the use of S_{11} which 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 an improved overall grid load balancing to 42% (as opposed to 31% in experiment 1).

Experiment No. 3

In experiment 3, the agent-based load-balancing mechanism is enabled for high-level resource discovery. This results in a new distribution of requests to the agents, where the more powerful platform receives more requests. 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 the grid resources such as S_1 and S_2 are only marginally improved by the GA scheduling when the workload is higher. No other agents show an improvement in local grid load balancing.

The experimental results in Table 3 are also illustrated in Figures 7, 8 and 9, which illustrate the effect on the performance metrics given in section 3.3. The curves indicate that different platforms exhibit

different trends when the system is configured with local and global 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, whilst others are indicated using grey lines. The curve for the overall grid is illustrated using a bold line.

Application Execution

In Figure 7, it is apparent that both the genetic algorithm and agent-based service discovery 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 experiments 1 and 2, and therefore the improvement of ε in experiments 2 and 3 is more significant. The results for 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 agent-based mechanism contributes more towards the improvement in application executions than the local schedulers.

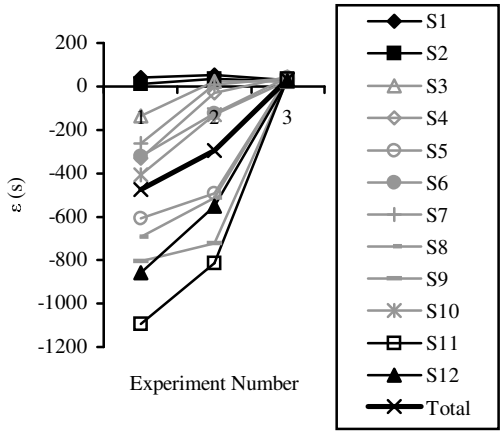

Figure 7. Case Study: Trends I for Experiment Results on Advance Times of Application Execution Completion ε

Resource Utilisation

The curves in Figure 8 illustrate similar trends to those of Figure 7. 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 agent-based 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 agent-based mechanism, which can dispatch more tasks to them.

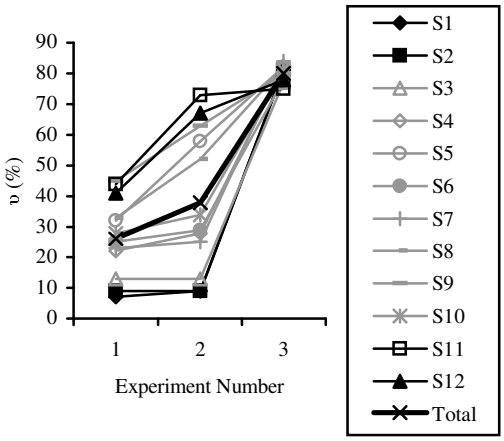

Figure 8. Case Study: Trends II for Experiment Results on Resource Utilisation rate υ

Load Balancing

Curves in Figure 9 demonstrate that local and global 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 Figures 7 and 8. In the second experiment when the GA scheduling is enabled, the load balancing of local grid resources are significantly improved. In the third experiment, when the agent-based 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 agents contribute more to global grid load balancing. The coupling of both is therefore a good choice to achieve grid load balancing at both local and global levels.

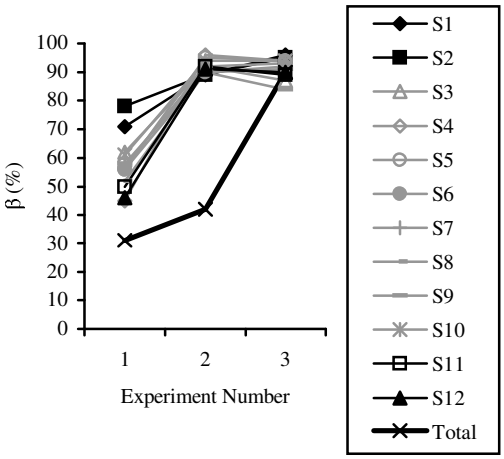

Figure 9. Case Study: Trends III for Experiment Results on Load Balancing Level β

5. Conclusions

This work addresses the problem of load balancing in a global grid environment. A GA-based scheduler has

been developed for fine-grained load balancing at the local level. This is then coupled with an agent-based mechanism that is applied to load balance at a higher level. Three experiments are carried out, with defined performance metrics used to measure the effect of grid load balancing. The experimental results demonstrate that the agent-based mechanism coupled with performance-driven task scheduling is suitable for grid load balancing. Load balancing can have large impact on both task executions and resource utilisation. In most situations, good load balancing results in task execution completing earlier with a better utilisation of grid resources. Higher level load-balancing mechanisms coupled with local level scheduling will provide a more effective solution to load balancing on the overall grid than the independent use of either approach.

The results presented in this paper will be compared with other similar systems. Future enhancement to the system will include the integration with other grid toolkits (e.g. Globus MDS and NWS). Experiments to test the scalability of the system will be carried out on a grid test-bed being built at Warwick.

Acknowledgements

This work is sponsored by grants from the NASA AMES Research Center (administered by USARDSG, contract No. N68171-01-C-9012), the EPSRC (contract No. GR/R47424/01), the EPSRC e-Science Core Programme (contract No. GR/S03058/01), and the NEC. The authors would like to express their gratitude to previous members of the group at Warwick, including Dr. S. C. Perry, Dr. J. S. Harper and Dr. D. J. Kerbyson, for their contribution to this work.

References

[1] A. Abraham, R. Buyya, and B. Nath, Nature's heuristics for scheduling jobs on computational grids, in "Proc. 8th IEEE International Conference on Advanced Computing and Communications", Cochin, India, 2000.

[2] D. Abramson, J. Giddy, and L. Kotler, High performance parametric modelling with Nimrod/G: killer application for the global grid? in "Proc. 14th International Parallel and Distributed Processing Symposium", Cancun, Mexico, 2000.

[3] D. Abramson, R. Sosic, J. Giddy, and B. Hall, Nimrod: a tool for performing parameterized simulations using distributed workstations, in "Proc. 4th IEEE International Symposium on High Performance Distributed Computing", Pentagon City, VA, USA, 1995.

[4] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, Application-level scheduling on distributed heterogeneous networks, in "Proc. 1996 Supercomputing", Pittsburgh, PA, USA, 1996.

[5] J. Cao, D. J. Kerbyson, E. Papaefstathiou, and G. R. Nudd, Performance modelling of parallel and distributed computing using PACE, in "Proc. 19th IEEE International Performance, Computing and Communication Conference", pp. 485-492, Phoenix, AZ, USA, 2000.

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

[7] 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* 5 (October 2001), 621-641.

[8] 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 International Symposium on Cluster Computing and the Grid", pp. 311-318, Brisbane, Australia, 2001.

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

[10] I. Foster and C. Kesselman, Globus: a metacomputing infrastructure toolkit, *Int. J. High Performance Computing Applications* 2 (1997), 115-128.

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

[12] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke, Condor-G: a computation management agent for multi-institutional grids, in "Proc. 10th IEEE Symposium on High Performance Distributed Computing", San Francisco, CA, USA, 2001.

[13] M. Litzkow, M. Livny, and M. Mutka, Condor – a hunter of idle workstations, in "Proc. 8th International Conference on Distributed Computing Systems", pp. 104-111, San Jose, CA, USA, 1988.

[14] H. Nakada, M. Sato, and S. Sekiguchi, Design and implementations of Ninf: towards a global computing infrastructure, *Future Generation Computing Systems* 5-6 (1999), 649-658.

[15] 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* 3(2000), 228-251.

[16] O. F. Rana, D. Bunford-Jones, D. W. Walker, M. Addis, M. Surridge, and K. Hawick, Resource discovery for dynamic clusters in computational grids, in "Proc. 10th IEEE Heterogeneous Computing Workshop", San Francisco, CA, USA, 2001.

[17] Y. Tanaka, Ninf-G: grid RPC system based on the Globus toolkit, in "The 2001 Globus Retreat", San Francisco, CA, USA, 2001.

[18] R. Wolski, N. T. Spring, and J. Hayes, The network weather service: a distributed resource performance forecasting service for metacomputing, *Future Generation Computing Systems* 5-6 (1999), 757-768.

[19] S. Zhou, LSF: load sharing in large-scale heterogeneous distributed systems, in "Proc. 1992 Workshop on Cluster Computing", 1992.

[20] A. Y. Zomaya and Y. The, Observations on using genetic algorithms for dynamic load-balancing, *IEEE Transactions on Parallel and Distributed Systems* 9 (September 2001), 899-911.