

# Local Grid Scheduling Techniques using Performance Prediction

D.P. Spooner\*, S.A. Jarvis\*, J. Cao†, S. Saini‡ and G.R. Nudd\*

\*High Performance Systems Group, Department of Computer Science,  
University of Warwick, Coventry, CV4 7AL, UK.  
Tel: +44 24 7652 2863 Fax: +44 24 7657 3024  
Email: dps@dcs.warwick.ac.uk

†C & C Research Laboratories, NEC Europe Ltd, St. Augustin, Germany.

‡NASA Ames Research Center, Moffett Field, California, USA.

## Abstract

The use of computational grids to provide an integrated computer platform, composed of differentiated and distributed systems, presents fundamental resource and workload management questions. Key services such as resource discovery, monitoring and scheduling are inherently more complicated in a grid environment where the resource pool is large, dynamic and architecturally diverse.

In this research, we approach the problem of grid workload management through the development of a multi-tiered scheduling architecture (TITAN) that employs a performance prediction system (PACE) and task distribution brokers to meet user-defined deadlines and improve resource usage efficiency. This paper focuses on the lowest tier which is responsible for *local* scheduling. By coupling application performance data with scheduling heuristics, the architecture is able to balance the processes of minimising run-to-completion time and processor idle time, whilst adhering to service deadlines on a per-task basis.

## 1 Introduction

It is anticipated that grids will develop to deliver high performance computing capabilities with flexible resource sharing to dynamic virtual organisations [1]. Essential to this growth is the development of fundamental infrastructure services that will integrate and manage large heterogeneous pools of computing resources, and offer them seamlessly to differentiated users. A key grid service is workload management, which involves distributing tasks and data over a selection of resources and coordinating communication and transfer systems.

Grid computing itself originated from a new computing infrastructure for scientific research and cooperation [2] and is rapidly becoming an established technology for large-scale resource sharing and distributed system integration [3]. The Open Grid Services Architecture (OGSA [4]) and its associated implementation, the Globus toolkit [5] is becoming a standard platform for grid service and application development, based upon web service protocols and open standards.

It is a common misconception that grid computing will provide general access to *free* resources and

high performance systems. This is not the case however, and Foster et al. predict in [1] that providers will charge for their services through the various grid accounting protocols that are currently in development [6]. In this consumer orientated model, quality-of-service (QoS) contracts must become an integral element of grid scheduling, where the perceived value of a service will steadily decrease as key service metrics such as deadline are not met.

In this research, a *performance prediction* system (PACE) is used to obtain parallel application performance data prior to run-time allowing resource requirements to be anticipated and deadlines considered. PACE [7] models consist of application and resource objects, with a parametric evaluation engine that provides expected execution traces and run-time predictions for different resource configurations. When coupled with a suitable scheduling algorithm, this approach can enhance deadline scheduling on heterogeneous multi-processor systems which motivates its application to grid computing.

The specific case of allocating independent tasks to multi-processor systems is examined in this work

where application performance models exist and the hardware has been characterised. The architecture, known as TITAN, adopts a conceptual multi-tiered approach to workload management, where standard Globus [5] grid protocols such as the Grid Index Information Service (GIIS [8]) and Globus Resource Allocation Management (GRAM [9]) are used at the top tier, service-providing brokers are used in the middle tier and localised scheduling systems are used at the lowest tier. This *local* and *global* partitioning provides a scalable platform for task and resource management.

This paper focuses on the lowest-level tier where PACE is used to estimate execution times for a particular task on a given set of architecture types prior to run-time. This is combined with an iterative heuristic (genetic) algorithm to select schedules and organise the task queue according to a given set of metrics. The rationale for selecting such an algorithm is that it is able to adapt to minor changes in the problem space (such as a host disconnecting, or a user submitting a task), and can be guided by means of a fitness function to improve task allocation according to multiple metrics.

The main contribution of this work is to apply an established performance prediction tool to grid scheduling by use of an iterative heuristic algorithm. This paper introduces the main theoretical considerations for the development of such a scheduling system, and then describes the implemented system with results based on an experimental workload. The organisation of this paper is as follows: section 2 introduces the workload management problem and how a performance prediction system can assist; in section 3, the algorithm used in the lowest tier is presented along with the coding scheme. Section 4 discusses the implementation and section 5 presents and evaluates experimental data; related work and conclusions are given in sections 6 and 7.

## 2 Grid Workload Management

While grid computing is not aimed specifically at the high performance computing (HPC) community, many organisations regard the grid as a means to deliver commodity computation that exceeds the capabilities of their current systems. Along with computationally powerful systems, grid providers of this type will need to employ effective management tools in order to steer codes to appropriate processing resources.

In a static environment, application performance may be improved through the tuning of an appropriate algorithm, the re-mapping of data, or the adjustment of communication behaviour all of which can be considered, to some extent, during development. In a dynamic grid setting, system-wide issues such

as resource configuration, user contention, node failure and congestion can incur a major impact on performance and are difficult factors to predict *a priori*, particularly when access to system information is restricted. Additionally, computational providers will invariably offer services to users with conflicting needs and requirements. Contract issues including deadline and response times must be balanced in order to meet respective service-level agreements.

It is therefore non-trivial for workload management systems to select suitable resources for a particular task given a varied, dynamic resource pool. The search space for this multi-parameter scheduling problem is large and not fully defined until run-time. As a result, a *just-in-time* approach is employed in this research where performance models of an application are evaluated as tasks are submitted allowing the system to consider run-time parameters prior to execution. The models are evaluated by local schedulers which are typically responsible for managing small to medium sized clusters of homogeneous processing nodes.

### 2.1 TITAN Architecture

TITAN adopts a loosely hierarchical structure consisting of workload managers, distribution brokers and Globus interoperability providers. The tiers map over conceptual divisions between group, organisation and inter-organisational boundaries as illustrated in Figure 1. In the current implementation, each distributed TITAN node can provide each or any of these services depending on configuration.

The purpose of the top tier is to provide Globus interoperability services. This includes passing GRAM requests to the distribution brokers and managing data that passes from the schedulers to the Globus information services. This represents the top of the TITAN chain, and hence information regarding schedulers and distribution brokers outside the organisation is achieved through the Globus Monitoring and Discovery Service (MDS). The middle tier consists of distribution broker systems which can manage subordinate brokers or schedulers. The brokers are based on the A4 advertising and discovery agent structure [10, 11]. The lowest tier consists of the schedulers that manage the physical computing resources.

### 2.2 Task Allocation

Tasks are submitted to the TITAN system via a text-based tool, a portal or (in development) GRAM. In each case, the request is delivered through the interoperability layer and received by a distribution broker. In this paper, each task must have an associated performance model which is currently stored in

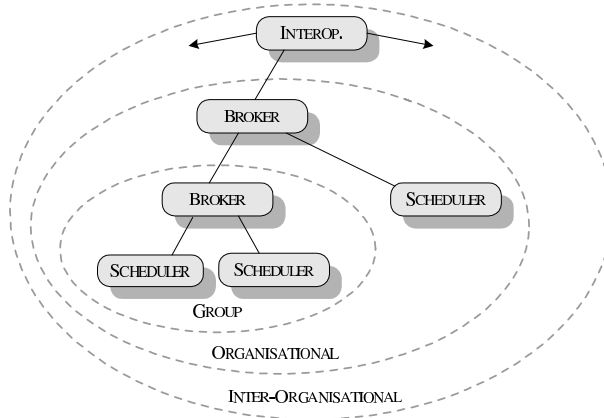


Figure 1: The TITAN architecture consisting of distribution brokers that communicate with other brokers and local schedulers to execute tasks in accordance with service policy constraints. An interoperability layer provides an interface to Globus permitting job submission to a particular broker.

a database indexed by task name, although a longer-term aim is to store details of the performance model in the Globus information service. When a task is submitted without a performance model, TITAN can be configured (by means of system policies) to move tasks to another broker, to a subset of resources in the cluster, or to the end or front of the current queue. While the lack of a performance model reduces TITAN to the functional equivalent of a batch scheduler, it is still able to make some decisions based on deadline, priority and resource requirement.

On receipt of an application request, the distribution broker interrogates its local scheduler (if one is associated) to ascertain whether the task can execute on the available resources and meet the user-specified deadline. The response is based upon evaluating a PACE model for the given task, the resource type, and by examining the current state of the schedule queue. If the scheduler can meet the user-defined deadline, the task is submitted to the local manager for scheduling. Otherwise, the broker attempts to locate a scheduler that can meet the requirements through discovery and advertisement mechanisms.

Each broker can maintain details of subordinate schedulers and make this data available via the GIIS (acts an information provider). This ‘advertisement’ strategy allows a foreign broker to interrogate the information service to determine whether a scheduler exists that can run a task and meet the deadline. If a suitable broker cannot be located by this method, the originating broker may initiate a ‘discovery’ process where it interrogates neighbouring brokers in the hierarchy. If these brokers are unable to meet the requirement, they pass on the request to their neighbours by means of a ‘discovery step’. If an ideal scheduler cannot be located within a preset number of discovery steps, the task request is either rejected or passed to a scheduler that can minimise the dead-

line failure depending on a task request parameter.

The balance of advertisement and discovery can be selected by a series of policies to suit the environment (and modified on-the-fly as required). Where task requests are relatively infrequent, the scheduling queues remain relatively static and so periodic advertisement of queue details to the information service is sufficient. Conversely, high load will result in many changes in the queue length and task mix. In this environment, brokers may need to ‘discover’ these details as necessary.

### 2.3 Schedule Representation

When a suitable scheduler is located, the task request is passed from the broker to the lowest tier whose objective is to organise tasks in a manner that satisfies contract policies and system metrics (such as the minimisation of execution time and idle time).

In this work, the localised scheduling problem is defined by the allocation of a group of independent tasks  $T = \{T_0, T_1, \dots, T_{n-1}\}$  to a network of hosts  $H = \{H_0, H_1, \dots, H_{m-1}\}$ . Tasks are run in a designated order  $\ell \in P(T)$  where  $P$  is the set of permutations.  $\ell_j$  defines the  $j^{\text{th}}$  task of  $\ell$  to execute where  $\forall \ell_j, \exists \beta_j \subseteq H, \beta_j \neq \emptyset$ , that is, each task in  $\ell$  has a suitable host architecture on which to run. To provide a compact representation of a schedule, a two dimensional coding scheme is utilised:

$$S_k = \begin{array}{c|cccc} & \ell_0 & \ell_1 & \dots & \ell_{n-1} \\ \hline H_0 & M_{0,0} & M_{0,1} & \dots & M_{0,n-1} \\ H_1 & M_{1,0} & M_{1,1} & \dots & M_{1,n-1} \\ \vdots & \vdots & \vdots & & \vdots \\ H_{m-1} & M_{m-1,0} & M_{m-1,1} & \dots & M_{m-1,n-1} \end{array} \quad (1)$$

where  $S_k$  is the  $k^{\text{th}}$  schedule in a set of schedules

$S = \{S_0, S_1, \dots, S_{p-1}\}$  and  $M_{ij}$  is the mapping of hosts to a particular application:

$$M_{ij} = \begin{cases} 1, & \text{if } H_i \in \beta_j \\ 0, & \text{if } H_i \notin \beta_j \end{cases} \quad (2)$$

For a set of identical processors, the predicted execution time for each task is a function,  $p_{ext}(\beta_j)_j$ , where  $\beta_j$  is the set of allocated resources for task  $\ell_j$ . Each arrangement of  $\beta_j$  is a *performance scenario* that will result in different run-time execution behaviour for a particular task. While codes exists that benefit from as many hosts as are available (i.e. embarrassingly parallel), most applications will experience degradation in performance as the communication costs outweigh the computation capability at a certain level of parallelism. Additional constraints such as memory size and hierarchy, node architecture and configuration will also effect application scalability. It is therefore important that the resource set ( $\beta_j$ ) is selected appropriately depending on the task involved. Performance prediction can assist in this selection via accurate modelling of scaling behaviour using software and hardware characterisation.

In the case where all the processing nodes are identical (as in the case of a cluster), the number of different performance scenarios is equal to the number of hosts. On a conventional system, it would be straightforward to formulate the function  $p_{ext}(\beta_j)_j$ , either by retaining historical data or by one-shot executions. However, in a heterogeneous grid environment, with architectural diversity and dynamic system properties, the number of performance scenarios increases rapidly. For any non-trivial resource network, it would be difficult to obtain  $p_{ext}(\beta_j)_j$  prior to scheduling.

## 2.4 Performance Prediction

The PACE system provides a method to obtain  $p_{ext}(\beta_j)$  dynamically, given an application model and suitable hardware descriptions. The hardware (resource) descriptions are generated when the resources are configured for use by TITAN, and the application models are generated prior to submission. The PACE evaluation combines these components with parameters to predict the execution time *just-in-time*. The speed of evaluation negates the need to store performance timings in a database, although TITAN uses a cache internally during its packing operation.

Assuming that a run-to-completion (non-overlap) strategy is adopted, TITAN is able to estimate the finish time for a particular task by summing the earliest possible start time plus the expected execution time, readily obtained by PACE from the

schedule representation  $S_k$ . The models are accurate for predicting the behavioural properties of computation applications on pre-modeled architectures. For applications whose internal structures have a low level of data-dependency, the performance models can be within 5% of the actual run-time. In this work, it is assumed that the performance model is accurate and that tasks will finish when expected – a strategy is discussed in section 5.2 to address the situation where a task overruns the expected completion time.

PACE models are modular, consisting of application, sub-task, parallel and resource objects. Application tools are provided that take C source code and generate sub-tasks that capture the serial components of the code by control flow graphs. It may be necessary to add loop and conditional probabilities to the sub-tasks where data cannot be identified via static analysis. The parallel object is developed to describe the parallel operation of the application. This can be reasonably straightforward for simple codes, and a library of templates exists for standard constructs. Applications that exhibit more complex parallel operations may require customisation. The sub-tasks and parallel objects are compiled from a performance specification language (PSL) and are linked together with an application object that represents the entry point of the model.

Resource tools are available to characterise the resource hardware through micro-benchmarking and modeling techniques for communication and memory hierarchies. The resultant resource objects are then used as inputs to an evaluation engine which takes the resource objects and parameters to produce predictive traces and an estimated execution time. This arrangement is illustrated in Figure 2.

The model forms a complex representation which can be used to establish prediction metrics for the execution of a particular application on a target architectural platform. The level of detail in the model can be chosen so as to improve accuracy and models are accumulated so that they can be reused. As the models are modular it is possible to evaluate  $p_{ext}(\beta_j)$  with differing host architectures, simply by replacing the resource object. This facilitates the application of PACE to heterogeneous environments as the application model can remain, for the most part, independent of the hardware. Examples of the use of PACE include performance optimisation of financial applications [12], on-the-fly performance analysis for application execution steering [13], and performance prediction using the Accelerated Strategic Computing Initiative (ASCI) demonstrator application Sweep3D [14].

Figure 3 demonstrates the connection between PACE and the scheduling system. The application tools are used by the users when the task is submitted

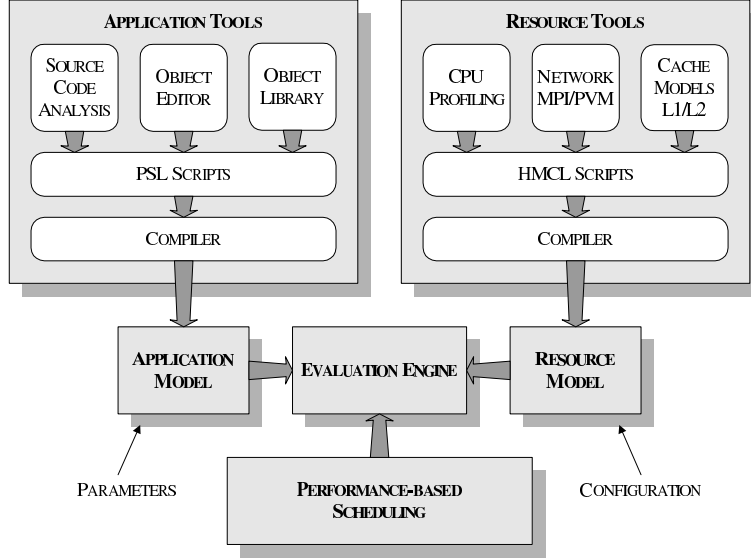


Figure 2: The PACE system consisting of application and resource tools that produce objects which form the input to a parametric evaluation engine.

if a model does not exist. With a good understanding of the code, model generation is reasonably straightforward. However, the PACE system is being developed to facilitate model generation for less well-known codes [15]. The scheduler uses the evaluation engine to identify expected execution run-time from the application models and from the resource models that represent the cluster or multi-processor system.

### 3 Localised Scheduling

In this work, a cluster is considered as a network of peer-to-peer hosts that are architecturally identical with similar communication costs between nodes. Scheduling issues are addressed at this level using task scheduling algorithms driven by PACE performance predictions. Three scheduling algorithms are developed in the section and are described below.

#### 3.1 Deadline Sort (DS)

When tasks are received by the scheduling system, it is possible to map the process to all possible nodes and determine the expected run-time using PACE. The expected end-time  $te_j$  for each task in the schedule queue can then be determined by summing the task start-time with the task's execution time, given by the following expression:

$$te_j = ts_j + p_{ext}(\beta_j)_j \quad (3)$$

The start time,  $ts_j$ , can be considered as the latest free time of all hosts mapped to the application. If all of the mapped hosts are idle, then  $ts_j$  is set to the current system time  $t$ , hence:

$$ts_j = \max_{\forall i, H_i \in \beta_j} \{tr_{ji}\} \quad (4)$$

where  $tr_{ji}$  is the *release time* for task  $j$  on host  $i$ . For tasks  $\ell_0 \dots \ell_{j-1}$  that execute on  $H_i$ , this is set to the end time of the application. Where there are no previous applications on this host, then this is set to the host release time,  $hr_i$ , which is the time when  $H_i$  is expected to become available. It is based upon tasks in the running queue and is set from  $te_j$  when tasks move out of the scheduling state. It is defined as:

$$tr_{ji} = \max_{\forall r, H_i \in \beta_r} \{TR_{jir}\} \quad (5)$$

$$TR_{jir} = \begin{cases} te_r, & \text{if } r < j \\ hr_i, & \text{if } r \geq j \end{cases} \quad (6)$$

Using this algorithm it is possible to sort tasks based on deadline and priority. However, no improvement in resource utilisation is made and so it is possible for tasks to 'over-utilise' a cluster. This results in a lower run-time for the task and prevents other tasks from using the cluster.

#### 3.2 Deadline Sort with Node Limitation (DS-NL)

The second approach extends the first by limiting the number of nodes allocated to a task based on predictive data. This can result in improved resource utilisation where the tasks use the number of nodes that result in the fastest run-time. Also, by selectively assigning the appropriate start node it is possible to parallelise two or more tasks if the number of nodes occupied is less than the sum of all nodes.

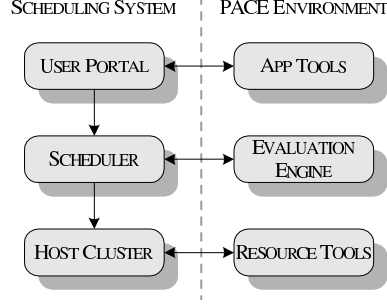


Figure 3: The connection between the scheduler and the PACE environment.

The problem with this algorithm is that a task can conceivably prevent another task from running at the expense of one overlapped node. In this instance, one solution may be to force one of the tasks to relinquish a node (if the task topology allows it). This is not a simple problem and the exploration of multiple scheduling solutions is an extension of the classical multi-processing (MS) scheduler problem, which is known to be intractable in the general case [16] and NP-hard for the case when  $m > 4$  where  $m$  is the number of machines.

### 3.3 Genetic Algorithm (GA)

The third approach uses a genetic algorithm to explore areas of the solution space to find good-quality schedules. Such algorithms are well-suited to search problems of this nature [17] and have been applied to a number of similar areas such as dynamic load balancing [18]. The approach used in this work generates a set of initial schedules, evaluates the schedules to obtain a measure of fitness, selects the most appropriate and combines them together using operators (crossover and mutation) to formulate a new set of solutions. This is repeated using the current schedule as a basis, rather than restarting the process, allowing the system to capitalise on the organisation that has occurred previously.

As with other iterative searching algorithms, this process is based upon the notion of a fitness function that provides a quantitative representation of how ‘good’ a solution is with respect to other solutions. In this manner, a set of solutions can be created and evaluated with respect to multiple metrics to isolate superior solutions. This ‘survival of the fittest’ approach maintains good solutions and penalises poor solutions in order to move towards a good-quality result.

This is achieved using an algorithm that selects suitable candidate schedules from a population of schedules in  $S$  and produces new schedules in  $S'$ . A fitness function is used to guide the selection of successful schedules by ejecting poor schedules from the system and maintaining good schedules in the

representation space.

#### 3.3.1 Fitness Function

The most significant factor in our fitness function is the make-span,  $\omega$ , which is defined as the time from the start of the first task to the end of the last task. It is calculated by evaluating the PACE function  $p_{ext}()_j$  for each host allocation and combining this with the end time of the task:

$$\omega = \max_{0 \leq j \leq n-1} \{te_j\} \quad (7)$$

Equation 7 represents the latest completion time of all the tasks where the aim of the process is to minimise this function with respect to the schedule. While schedules can be evaluated on make-span alone, in a real-time system, idle time must also be considered to ensure that unnecessary space is removed from the start of the schedule queue. TITAN employs an additional function (8) that penalises early idle time more harshly than later idle time, using the following expression:

$$\Gamma_k = \omega_k \sum_i \left( \frac{\Delta T_i^{\text{idle}}}{\omega_k^2} (2\omega_k - 2T_i^{\text{idle}} - \Delta T_i^{\text{idle}}) \right) \quad (8)$$

where  $T_i^{\text{idle}}$  is the start of the idle time, and  $\Delta T_i^{\text{idle}}$  is the size of idle time space which can be calculated as the difference between the end time of the task and the start time of the next task on the same host. This idle weighting function is therefore a decrease from 100% weighting at  $(T_i = 0, \Delta T_i = \omega)$  to 0% weighting at  $(T_i = \omega, \Delta T_i = 0)$ . The reason for penalising early idle time is that late idle time has less impact on the final solution as:

1. the algorithm has more time to eliminate the pocket;
2. new tasks may appear that fill the gap.

A linear contract penalty is also evaluated which can be used to ensure that the deadline time for a task,  $d_j$ , is greater than the expected end-time of the task.

$$\theta_k = \sum_{j=0}^{n-1} \max \{(te_j - d_j), 0\} \quad (9)$$

The three fitness functions are combined with preset weights to formulate an overall fitness function (10). Using these weights, it is possible to change the behaviour of the scheduler *on-the-fly*, allowing the scheduler to prioritise on deadline or compress on make-span, for example.

$$f_k = \frac{(W^i * \Gamma_k) + (W^m * \omega_k) + (W^c * \theta_k)}{W^i + W^m + W^c} \quad (10)$$

The fitness value is normalised to a cost function using a dynamic scaling technique, and then used as the basis for the selection function. Schedules with a higher fitness value are more likely to be selected than a schedule with a fitness approaching 0.

### 3.3.2 Crossover / Mutation

The selected schedules are subject to two algorithmic operators to create a new solution set, namely ‘crossover’ and ‘mutation’. The purpose of crossover is to maintain the qualities of a solution, whilst exploring a new region of the problem space. Mutation adds diversity to the solution set.

As the schedules are represented as a two dimensional string that consists of a task ordering part (which is q-ary coded) and a binary constituent (which is binary coded), the genetic operators have to be different for the task order and host mappings. Each operator acts on a pair of schedules, and is applied to the entire solution set.

$$S \xrightarrow{\text{crossover}} S' \xrightarrow{\text{mutation}} S'' \quad (11)$$

The task crossover process consists of merging a portion of the task order with the remaining portion of a second task order, and then re-ordering to eliminate illegal orders. The mutation process involves swapping random task orders. For example, two task orders can be represented as follows:

$$\begin{aligned} \ell_A &= [T_1, T_3, T_5, T_2, T_0, T_6, T_4, T_7] \\ \ell_B &= [T_4, T_5, T_2, T_1, T_6, T_7, T_0, T_3] \end{aligned}$$

$\ell_A$  and  $\ell_B$  can be combined at a random point to produce a new schedule task list  $\ell'$ , where  $\ell_{A,n}$  denotes the  $n^{\text{th}}$  element of  $\ell_A$ . Task duplications are then removed by undoing the crossover operation for the repeated task, as illustrated in the following expression where  $(T_3)$  denotes a task duplication which is resolved by setting the first  $(T_3)$  to the first unused value of  $\ell_B$

$$\begin{aligned} \ell' &= [\ell_{A.0}, \ell_{A.1}, \ell_{A.2}, \ell_{B.3}, \ell_{B.4}, \ell_{B.5}, \ell_{B.6}] \\ &= [T_1, (T_3), T_5, T_2, T_6, T_7, T_0, (T_3)] \\ &= [T_1, T_4, T_5, T_2, T_6, T_7, T_0, T_3] \end{aligned} \quad (12)$$

Random changes are then applied to  $\ell'$  to give the mutated representation  $\ell''$ , illustrated with the bracketed tasks in the following expression:

$$\begin{aligned} \ell'' &= [T_1, T_4, T_5, T_2, T_6, (T_7), T_0, (T_3)] \\ &= [T_1, T_4, T_5, T_2, T_6, T_3, T_0, T_7] \end{aligned} \quad (13)$$

The host map crossover process merges the host component of the two schedules together, using a simpler binary copy. The mutation process randomly toggles bits in the hostmap, while ensuring that topology rules have not been broken (such as trying to run the task on zero hosts for example). Expression 14 represents a full schedule of 8 tasks and 5 hosts.

$$\begin{aligned} \ell'' &= [T_1, T_4, T_5, T_2, T_6, T_3, T_0, T_7] \\ M'' &= \begin{bmatrix} [1, 1, 0, 1, 1], & [1, 1, 0, 0, 0], \\ [0, 0, 1, 0, 0], & [0, 0, 0, 1, 0], \\ [0, 0, 0, 0, 1], & [1, 1, 0, 0, 1], \\ [1, 1, 1, 1, 1], & [1, 0, 1, 0, 1]. \end{bmatrix} \end{aligned} \quad (14)$$

Figure 4 depicts a visualisation of this representation. The tasks are moved as close to the bottom of the schedule as possible, which can result in a task that has no host dependencies being run prior to a task earlier in the task order. This is illustrated with  $T_5$  being scheduled before  $T_4$  despite being further on in the task order.

## 4 Implementation

The techniques presented in the previous section have been developed into a working system for scheduling parallel tasks over a heterogeneous network of resources. The GA algorithm introduced in section 3 forms the centre point of the localised workload managers and is responsible for selecting, creating and evaluating new schedules. Connection to the PACE system is achieved by a remote evaluation library, with a performance cache to store timings for subsequent re-use.

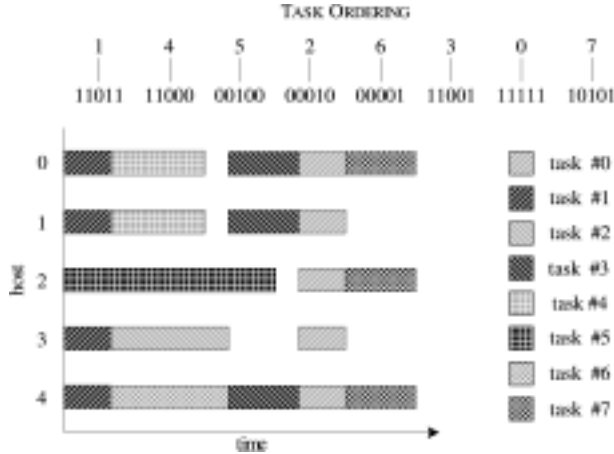


Figure 4: Visualisation of the schedule given in expression 14. Task  $T_5$  is run earlier than  $T_4$  as it requires  $H_2$  which is not used by either  $T_1$  or  $T_4$ .

#### 4.1 Workload Management Process

The core data type is the *Task*. Tasks are submitted to the workload manager from the distribution brokers, and enter a pool of Task objects. The scheduler maintains continuous visibility of this pool and applies the strategies described earlier to form solution sets. The scheduler also accepts host information from a second pool which is updated by an on-line host gathering and statistics module. Currently, this information is gained from UNIX ‘uptime’ and ‘sar’ commands, although a small agent is being developed to remove the operating system dependence that this approach involves. The information is to check the load, status and idle-time of the hosts in the cluster. It is therefore possible for TITAN to utilise hosts as ‘dual-mode’ machines which are only used for scheduling when they have been idle for a preset period of time. As this level of resource is below the Globus layers, the MDS cannot be used for statistics gathering.

At each iteration, the best solution is compared with an overall fittest solution. This allows the scheduler to maintain a copy of the best schedule found to date, whilst allowing the process to explore other regions of the problem space. If a significant change occurs in either the host or task pools, then this schedule is deemed ‘stale’ and replaced by the fittest schedule in the current generation.

Periodically, the scheduler examines the bottom of the schedule and if there are any tasks to run, will move them into the run queue and remove them from the scheduling queue. Figure 5 provides a summary view of this system.

#### 4.2 GA Performance

In order to test the GA operation, simulations were run in ‘test’ mode where tasks are not moved into the

executing state. This allows the operation of the algorithms to be tested and tuned. In this simulation, a workload of 32 tasks was created for a 16 node cluster. Each task was selected from a set of five codes with different performance scaling characteristics as illustrated in Figure 6. Each of the codes have associated PACE performance data which can be used to provide the scaling curves. Details of model generation can be found for the Sweep3D model in [19].

In the first experiment, the test schedule is evaluated to 6000 generations with 3 different population sizes:  $S = \{S_0, S_1, \dots, S_{19}\}$ ,  $\{S_0, S_1, \dots, S_{39}\}$ ,  $\{S_0, S_1, \dots, S_{59}\}$ .

The first 2000 iterations are shown in Figure 7 which illustrates the classical GA properties: namely rapid convergence to (an often) sub-optimal solution. Reducing the population size constrains the diversity which can have a direct effect on identifying a good quality solution. However, increasing the population has computation and storage implications, limiting the number of iterations that can be achieved between tasks. An approximate cost of this technique is given by:

$$t_{eval} = N_p N_t [C_{ga} + C_{pace}] \quad (15)$$

where  $N_p$  is the number schedules in the population,  $N_t$  is the number of tasks in the scheduling queue, and  $C_{ga}$  and  $C_{pace}$  are the computational cost of executing the GA operators and the PACE evaluation engine respectively. Hence, reducing the population size permits more iterations/sec which can move more rapidly toward a solution. During experimentation, it was found that the difference between population sizes 40 and 60 were minimal with a significant saving in run-time. With a population of 40, the scheduler (running on a standalone node with a 800Mhz Pentium III) can deliver 100 iterations per/sec assuming that the PACE data has been

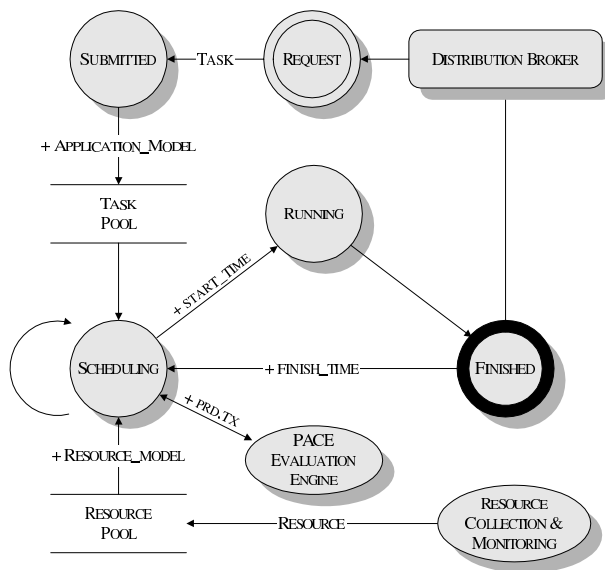


Figure 5: Diagram that details the key system states.

cached (which typically occurs after the initial few iterations).

In the second experiment, the weighting function of (10) is demonstrated by introducing synthetic deadlines at approximately 1000 generations. Figure 8 contains the normalised graphs of the make-span contribution, the deadline contribution and the global fitness. The idle-time weight is neglected in this experiment as it has a less visible effect in the tests, and is more useful when tasks are executing.

Initially, the unsorted schedule results in a high make-span and associated make-span contribution to the overall global fitness. As the iterations continue, the make-span is reduced and the population fitness improves. At generation 600, the imminent deadline constraints start to effect the global fitness, although make-span is still the dominant factor. As the tasks are not removed from the queue in this ex-

periment, the deadline metric becomes progressively dominant and at 1000 generations the scheduler becomes deadline-driven as opposed to makespan-driven, demonstrating how multiple parameters can be considered dynamically.

## 5 Evaluation

To schedule and run tasks, TITAN is started in 'execution' mode (as opposed to the 'test' mode described previously). The same 32 tasks assigned in the previous section were used with deadlines set cyclically between 5 and 11 minutes, and priorities assigned between 1 and 5 where 1 is the most important. Each task was submitted to the TITAN scheduler using a proprietary text-based tool. Work on adding GRAM interoperability is in progress and

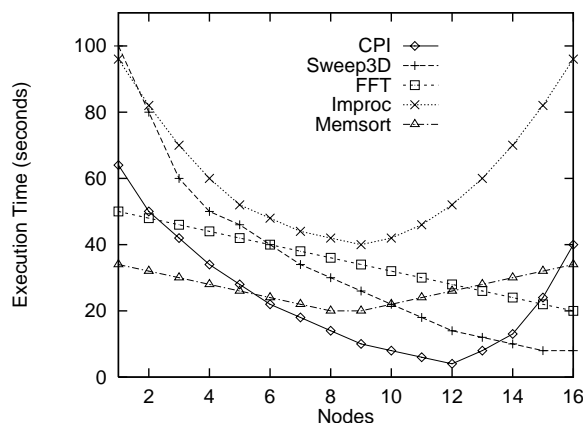


Figure 6: Scaling characteristics for CPI, the Sweep3D ACSI demonstrator code, an FFT routine, an image processing code and a bitonic memory sort – based on a resource timings for a cluster of Pentium III 800Mhz based systems connected over a Fast Ethernet network.

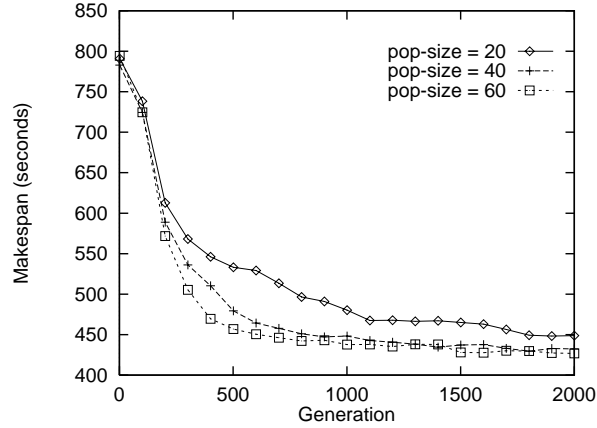


Figure 7: Make-span comparison for three different population sizes.

it will allow the scheduler to be utilised from Globus. In addition, the scheduler will make key service data available via the information services.

## 5.1 Run-time Evaluation

In the first experiment, the three scheduling algorithms are compared by make-span. They are set against a simple *first-come-first-served* (FC-FS) algorithm which performs no adjustment other than sorting by priority. Figure 9 illustrates the number of tasks completed over time. In this test, the FC-FS and the deadline sorting demonstrate the worst-case make-span as they are unable to reconfigure tasks to improve resource usage. The approach of limiting the tasks to the maximum required nodes results in an expected drop in make-span. However, this operation is critically related to the number of nodes, and hence the GA algorithm, which is adaptable, is able to improve on this by 20% by making slight modifications that can fill the gaps left by the node limitation technique.

In each of these cases, it is possible to calculate the average advance time ( $A_T$ ) which gives an indication of how efficiently deadlines are being met. This is calculated as follows:

$$A_T = \frac{\sum d_j - TE_j}{n} \quad (16)$$

where  $d_j$  is the user-defined deadline,  $TE_j$  is the actual end-time of the application  $j$ , and  $n$  is the number of tasks in the queue. Table ?? provides the average advance time for each approach.

The negative values of FC-FS and deadline sort indicate that many of the deadlines were not met, indeed on average both approaches missed their respective deadlines by over two minutes.

## 5.2 Dynamic Evaluation

The advantage of the GA approach over the node limitation algorithm is that it is not so sensitive to the number of processing nodes. This is especially apparent if an event occurs such as a host going offline. In the final simulation, 4 of the 16 processors

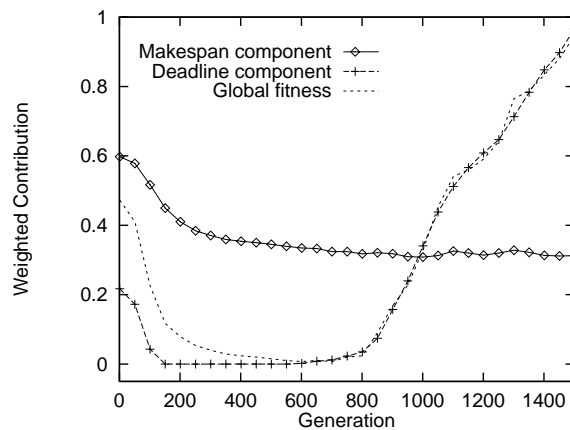


Figure 8: The contribution of makespan and deadline to the global fitness of the population (lower is fitter). At 1000 generations, the scheduler is more influenced by deadline than makespan.

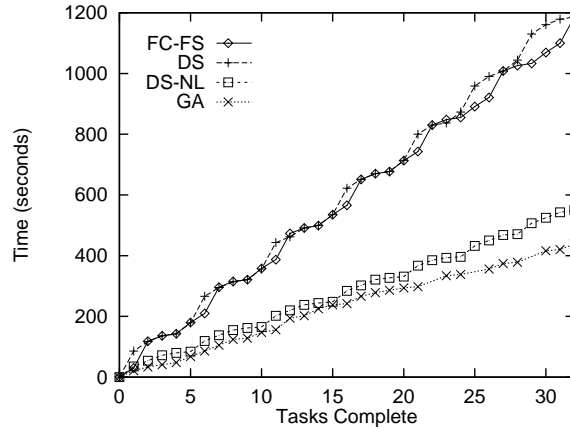


Figure 9: The GA algorithm results in the most rapid execution of the scheduler queue, compared with first-come first-served (FC-FS), deadline sort (DS) and deadline sort with node limitation (DS-NL).

are removed after approximately 60 seconds of run-time. The node-limited algorithm takes an immediate hit which pushes the make-span up accordingly. Without any means to work around the gaps, all the tasks are pushed back and it takes over 2 minutes for the make-span to fall back previous levels. The GA, which has a lower make-span from the outset, copes far better with the gap, recovering the make-span in less than a minute.

In section 2.4 it stated that the scheduler makes the assumption that PACE is 100% accurate. Although tests have yet to be run, it is felt that the GA will cope well with deviations from expected run-time in much the same manner as when hosts go off-line, reorganising the schedule queue where necessary.

## 6 Related Work

There are a number of grid projects that address various workload and resource management issues, including Nimrod [20], NetSolve [21], AppLeS [22], Ninf [23], Condor [24], LSF [25] and the Grid Resource Broker (GRB) [26] project. Some of these works, including Nimrod, Ninf and AppLeS are similar to TITAN/PACE in that performance models are employed to enhance resource utilisation through effective allocation. Nimrod, for example, has been developed specifically to assist in virtual laboratory work and is able to orchestrate distributed experiments using a parametric evaluation engine and declarative performance modeling language. It also utilises a scheduling heuristic [27] to allocate resources. Nimrod/G [28] extends Nimrod with a grid-aware version that uses grid protocols to perform automatic discovery of resources.

A different approach is taken by AppLeS by placing the onus on the user to provide the scheduling functionality which is embedded into the ap-

plication. Agents that contain both static and dynamic performance information query the Network Weather Service [29] at the time of execution and use algorithms to make a selection whether resources are suitable candidates or not. Ninf also utilises the Network Weather Service and is based on performance evaluation techniques. The approach of NetSolve to provide remote computation facilities is through the selection of the highest performing resources in a network. Unlike TITAN/PACE however, modifications to the application source code are required in order to utilise NetSolve's services.

Other related grid projects include batch queuing systems that provide system-level cluster scheduling including Condor, Condor/G [30] and LSF. These system do not, however, necessarily aim to find the best scheduling solution, while TITAN has been developed with this in mind. Both Condor and LSF provide facilities for managing large, high throughput grid resources and Condor, in particular, integrates tightly with Globus. TITAN requires further development to achieve full Globus integration. Other scheduling tools that integrate with Globus include GRB which provides access to Globus services using a web-based portal, permitting location-transparent scheduling of tasks. It is a more general tool than TITAN using resource finders and matchmakers based on resource and user characteristics as opposed to performance models.

As grid development continues, the requirement to consider performance issues will become increasingly important. While effective utilisation and allocation is desirable for system and resource providers, performance characterisation provides the *user* with realisable deadline and priority parameters. It is anticipated that future implementations of Globus will feature reservation services in which performance prediction will be crucially important to prevent wasteful allocations and meaningless user metrics (i.e. such as non-fulfilable deadlines). Considera-

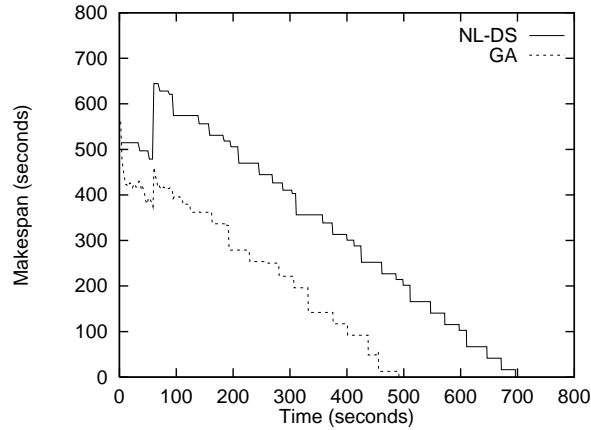


Figure 10: Make-span of schedule queue over time. After 1 minute, 4 hosts are taken off-line resulting a schedule reorganisation. The GA copes well with this incremental change.

tion of performance issues at the ‘middleware’ level, the approach taken by many of the works above and TITAN, will help resolve these issues. The approach of TITAN is to utilise an established and flexible performance characterisation system (PACE) with a GA-based algorithm. While PACE provides true hardware and software separation, which is ideal for heterogenous grid environments, it would be possible for TITAN to utilise other performance tools including POEMS [?] or analytical models such as [?].

The use of genetic algorithms (GA) in dynamic multi-processing scheduling systems has been explored previously [31, 32] where they have been successfully applied to the minimisation of schedule length (make-span) for dissimilar tasks. TITAN augments the traditional GA operations of crossover and mutation with application-centric data obtained through PACE model evaluation including the ability to reject schedules that over-utilise resources. As PACE is able to produce accurate results rapidly, it can be used in iterative procedures (although some caching is required). Evaluating PACE models prior to execution also allows the system to consider variables that are not available during the traditional performance analysis stage which often require recomposition of the model.

## 7 Conclusions

The work presented in this paper is concerned with improving grid workload management using a multi-tiered framework based on Globus providers, distribution brokers and local schedulers. This paper has focused primarily on the iterative heuristic algorithm and performance prediction techniques that have been developed for the local schedulers. Current work is focussed on fully implementing operability with Globus to enable performance-based ‘global’ scheduling in addition to ‘local’ scheduling.

This work was initially based on Globus 2, and is now being updated to integrate with OGSA and associated services.

A test suite has been implemented to tune the algorithm parameters and the PACE models. The results demonstrate that the scheduler converges quickly to a suitable schedule pattern, and that it balances the three functions of idle time, make span and the quality-of-service metric deadline time.

The PACE system provides an effective mechanism for determining general application performance through a characterisation technique. The resultant models can be used for performance studies, design work and code-porting activities. In this work, they have been adapted for use with a GA-based scheduler for provide the basis for performance-based middleware for general applications.

Further work will examine the relationship between low-level scheduling and the middle tier of workload management and top tier of wide area management. It is also possible to extend the system to manage flows of work as groups of related tasks. It is envisaged that this framework will develop to provide complimentary services to existing grid infrastructures (i.e. Globus) using performance prediction and service brokers to meet quality demands for grid-aware applications.

## Acknowledgments

This work is sponsored in part by grants from the NASA AMES Research Center (administrated by USARDSG, contract no. N68171-01-C-9012) and the EPSRC (contract no. GR/R47424/01). The authors would also like to express their gratitude to Darren Kerbyson and Stuart Perry for their contribution to this work.

## References

- [1] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organisations. *Int. J. of Supercomputing Applications*, 2001.
- [2] I. Foster and C. Kesselman. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1998.
- [3] I. Foster, C. Kesselman, J.M. Nick, and S. Tuecke. Grid services for distributed system integration. *IEEE Computer*, 35(6):37–46, 2002.
- [4] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. *Open Grid Service Infrastructure WG, Global Grid Forum*, Jun 2002.
- [5] I. Foster and C. Kesselman. Globus: A Meta-computing Infrastructure Toolkit. *Int. J. of Supercomputer Applications*, 11(2):115–128, 1997.
- [6] Global Grid Forum. <http://www.globalgridforum.org>.
- [7] 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. of High Performance Computing Applications, Special Issues on Performance Modelling*, 14(3):228–251, 2000.
- [8] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman. Grid Information Services for Distributed Resource Sharing. In *Proc. of the 10th IEEE International Symposium on High-Performance Distributed Computing (HPDC-10)*, 2001.
- [9] The Globus Toolkit. <http://www.globus.org/toolkit>.
- [10] J. Cao, D.J. Kerbyson, and G.R. Nudd. High Performance Service Discovery in Large-scale Multi-agent and Mobile-agent Systems. *Int. J. of Software Engineering and Knowledge Engineering, Special Issue on Multi-Agent Systems and Mobile Agents*, 11(5):621–641, 2001.
- [11] 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(2):135–148, 2002.
- [12] S.C. Perry, R.H. Grimwood, D.J. Kerbyson, E. papaefstathiou, and G.R. Nudd. Performance Optimisation of Financial Option Calculations. *Parallel Computing*, 26(5):623–639, 2000.
- [13] D.J. Kerbyson, E. Papaefstathiou, and G.R. Nudd. Application Execution Steering using On-the-fly Performance Prediction. *High Performance Computing and Networking, Lecture Notes in Computer Science*, 1401:718–727, 1998.
- [14] J. Cao, D.J. Kerbyson, E. Papaefstathiou, and G.R. Nudd. Modelling of ASCI High Performance Applications using PACE. *Proc. UK Performance Engineering Workshop (UKPEW'99)*, pages 413–424, 1998.
- [15] J.D. Turner, D.P. Spooner, S.A. Jarvis, D.N. Dillenberger, and G.R. Nudd. A Transaction Definition Language for Java Application Response Measurement. In *Proc. of 27th Int. Conf. on Technology Management and Performance Evaluation of Enterprise-Wide Information Systems (CMG2001)*, Anaheim, California, USA, December 2001.
- [16] J. Du and J. Leung. Complexity of Scheduling Parallel Task Systems. *SIAM J. on Discrete Mathematics*, November 1989.
- [17] D. Dumitrescu, B. Lazzerini, L.C. Jain, and A. Dumitrescu. *Evolutionary Computation*. CRC Press, 2000. ISBN: 0–84–930588–8.
- [18] A.Y. Zomaya and Y.H. Teh. Observations on Using Genetic Algorithms for Dynamic Load-Balancing. *IEEE Transactions on Parallel and Distributed Systems*, 12(9):899–911, 2001.
- [19] K.R. Koch, R.S. Baker, and R.E. Alcouffe. Solution of the First-Order Form of the 3-D Discrete Ordinates Equation on a Massively Parallel Processor. *Trans. of the Amer. Nuc. Soc.*, 65(108), 1992.
- [20] 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. of 4th Int. Conf. on High Performance Computing*, Asia-Pacific Region, Beijing, China, 2000.
- [21] H. Casanova and J. Dongarra. NetSolve: A Network Server for Solving Computational Science Problems. *Int. J. of Supercomputing Applications and HPC*, 11(3), 1997.
- [22] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao. Application-level Scheduling on

- Distributed Heterogeneous Networks. *Proc. of Supercomputing*, 1996.
- [23] A. Takefusa, S. Matsuoka, H. Nakada, K. Aida, and U. Nagashima. Overview of a Performance Evaluation System for Global Computing Scheduling Algorithms. In *Proc. of 8th IEEE Int. Symp. on High Performance Distributed Computing*, pages 97–104, 1999.
- [24] M. Litzkow, M. Livny, and M. Mutka. Condor – A Hunter of Idle Workstations. In *Proc. of 8th Int. Conf. on Distributed Computing Systems (IPDCS88)*, pages 104–111, 1988.
- [25] S. Zhou. LSF: Load Sharing in Large-scale Heterogenous Distributed Systems. In *Proc. of 1992 Workshop on Cluster Computing*, 1992.
- [26] G. Aloisio, M. Cafaro, E. Blasi, and I. Epicoco. The Grid Resource Broker, a Ubiquitous Grid Computing Framework. *Scientific Programming : Special issue on Grid Computing*, 10(2), 2002.
- [27] A. Abraham, R. Buyya, and B. Nath. Nature’s Heuristics for Scheduling Jobs on Computational Grids. In *Proc. of 8th IEEE Int. Conf. on Advanced Computing and Communications*, Cochin, India, 2000.
- [28] R. Buyya, D. Abramson, and J. Giddy. Nimrod-G resource broker for service-oriented grid computing. *IEEE Distributed Systems Online*, 2(7), 2001.
- [29] R. Wolski, N.T. Spring, and J. Haye. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computing Systems*, 1999.
- [30] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids. In *Proc. of the 10th IEEE Symposium on High Performance Distributed Computing (HPDC10)*, 2001.
- [31] E.S. Hou, N. Ansari, and H. Ren. Genetic Algorithm for Multiprocessor Scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 5(2):113–120, 1994.
- [32] M.D. Kidwell and D.J. Cook. Genetic Algorithm for Dynamic Task Scheduling. In *Proc. of IEEE 13th Annual International Phoenix Conference on Computings and Communications*, pages 61–67, 1994.

## Appendix A : TITAN Screen Shots



Figure 11: Visualisation of the schedule before GA operation.

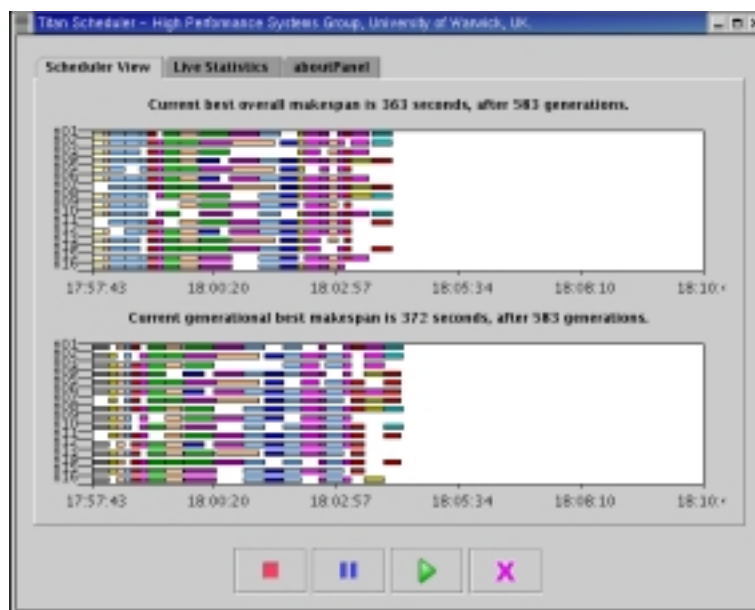


Figure 12: Visualisation of the schedule after 300 iterations.