International Journal of Computing and Network Technology

@ 2013 UOB
SPC, University of Bahrain

A Novel Workload Allocation Strategy for Batch Jobs

A. Shenfield[1] and P. J. Fleming[2]

[1]School of Engineering, Manchester Metropolitan University, Manchester, UK
Email Address: a.shenfield@mmu.ac.uk
[2]Department of Automatic Control and Systems Engineering, The University of Sheffield, Sheffield, UK
Email Address: p.fleming@sheffield.ac.uk

Received: 12 Nov. 2012, Revised: 20 Nov. 2012; Accepted: 13 Dec. 2012

Abstract: The distribution of computational tasks across a diverse set of geographically distributed heterogeneous resources is a critical issue in the realisation of true computational grids. Conventionally, workload allocation algorithms are divided into *static* and *dynamic* approaches. Whilst dynamic approaches frequently outperform static schemes, they usually require the collection and processing of detailed system information at frequent intervals - a task that can be both time consuming and unreliable in the real-world. This paper introduces a novel workload allocation algorithm for optimally distributing the workload produced by the arrival of batches of jobs. Results show that, for the arrival of batches of jobs, this workload allocation algorithm outperforms other commonly used algorithms in the static case. A hybrid scheduling approach (using this workload allocation algorithm), where information about the speed of computational resources is inferred from previously completed jobs, is then introduced and the efficiency of this approach demonstrated using a real world computational grid. These results are compared to the same workload allocation algorithm used in the static case and it can be seen that this hybrid approach comprehensively outperforms the static approach.

Keywords: Workload Allocation, Adaptive Resources Scheduling, Grid Computing

I. Introduction

Computational grids are becoming a widespread high performance computing platform for scientific applications due to their ability to integrate existing heterogeneous computational resources across multiple geographically distributed sites. However, a fundamental issue in the performance of these grid enabled scientific applications is the ability to schedule and run jobs across these multiple diverse computational resources in an optimal way. Whilst several workload allocation strategies exist in the literature (see Section II), they often either require complex dynamic information about the current state of the system or assume that jobs arrive at the grid resources individually (whereas many scientific applications require the parallel processing of batches of many independent tasks - such as evolutionary optimisation methods, parameter sweeps and finite element methods).

This paper introduces a novel static workload allocation algorithm in section 0 that distributes batches of independent computational tasks in an optimal way and, in section 0, shows that this algorithm performs favourably when compared to common benchmarks from the literature. Then, in section VI, this optimal static algorithm is used as the basis of a hybrid approach to job scheduling that overcomes the need for the time consuming collection and processing of complex system information. Results from a real-world computational grid are presented in section 0-BVI.b showing that the proposed low-cost hybrid approach significantly outperforms the same workload allocation algorithm used in the static case.

II. BACKGROUND

Workload allocation strategies can be classified as either *static* or *dynamic* [1]-[3]. Static workload allocation schemes use *a priori* information to make scheduling decisions, whilst dynamic workload allocation schemes use runtime information about the state of the system in making decisions. Dynamic schemes often outperform static schemes because they can adapt to changes in workload and system state [4], [5]. However, dynamic workload allocation schemes have a much higher overhead because they have to collect and process system information at frequent intervals. In a heterogeneous, distributed environment (such as a computational grid) the collection of this dynamic system information is often time-consuming, unreliable and inconsistent [6]. [7] argue that, for some applications, the performance gain from a good low-cost static algorithm may be more beneficial than the overheads involved in collecting detailed dynamic information.

Workload allocation policies for distributed systems are often analysed in terms of queueing theory (see [8] for an introduction to queueing theory). [7] have investigated the optimisation of static job schedules in networks of heterogeneous[1] workstations. They modelled each computer in their system as an *M/M/1* queue with arrival rate λ, baseline service rate μ, and relative processing speed s_i. Their aim was to find the optimal allocation of workload, α_i, for each of the n nodes so as to minimise the mean response time of the system (shown in (1) below).

$$\overline{T}_{sys} = \sum_{i=1}^{n} \alpha_i \frac{1}{s_i\mu - \alpha_i\lambda} \qquad (1)$$

[7] showed that this static optimised mean response time (ORT) workload allocation algorithm significantly outperformed the benchmark weighted workload allocation scheme (where workload was allocated to nodes in proportion to their processing speed). They used a discrete event simulator to evaluate the performance of the workload allocation algorithms and showed that their optimised mean response time (ORT) static workload allocation scheme outperformed the weighted workload allocation scheme. This confirmed earlier results by [9] who found that, at low and moderate levels of system load, it is more efficient to allocate a disproportionately high share of workload to the more powerful nodes.

[10] have also used queueing theory to investigate optimal static workload allocations by extending the work in [7] to multi-clusters (where each node in the system is a homogeneous multi-cluster consisting of *m* processing nodes). Each system node is therefore modelled as an *M/M/m* queue, and two objectives for workload allocation are examined: optimising the mean response time for non-real-time jobs in the system, and optimising the mean miss rate[2] (OMR) for jobs with soft real-time constraints. The performance of the workload allocation schemes developed in [10] has been evaluated using discrete event simulation, and both optimal strategies perform well with respect to their objectives when compared to a weighted workload allocation scheme.

[5] compared two static workload allocation policies with six dynamic policies for several systems under a variety of loads. The evaluation of the different workload allocation policies was performed using

[1] Heterogeneous in the context of this paper refers to differences in computational capability rather than differences in CPU architecture or operating system

[2] For Quality-of-Service (QoS) demanding jobs, the mean miss rate is the average number of jobs that fail to meet the required QoS level. The profits under a Service Level Agreement (SLA) are maximised if all the jobs meet their QoS demands.

simulation techniques and, in general, the dynamic policies outperformed the static schemes. The exception to this was when the collection of system information proved to be expensive. In this case [5] recommended the use of a static workload allocation policy (such as the one developed in [7] or [10]).

III. MODELLING OUR SYSTEM

Computational grids such as the White Rose Grid[3] can be modelled as a network of heterogeneous compute resources[4], $\{r_1, r_2, ..., r_n\}$, with each resource having service rate $\{\mu_1, \mu_2, ..., \mu_n\}$ (see Fig. 1). The mean arrival rate of jobs into the system is λ and the task of the global scheduler in Fig. 1 is to allocate a fraction of the workload, α_i, to each resource according to the specified scheduling scheme.

This paper will focus on the bulk arrival of jobs into the system (i.e. where each arrival corresponds to a group of *k* jobs), since many complex applications involve the submission of groups of independent tasks for execution in parallel (for example, computational fluid dynamics simulations and parameter sweeps both involve the parallel processing of multiple computational tasks). Many of the static workload allocation algorithms proposed in the literature focus on computational tasks arriving at the global scheduler individually [7], [10], an approach that is suboptimal for the kind of batch processing associated with these applications. This is because these workload allocation algorithms assume that individual tasks arrive according to an exponential distribution and, since batches of jobs arrive together, this assumption no longer holds.

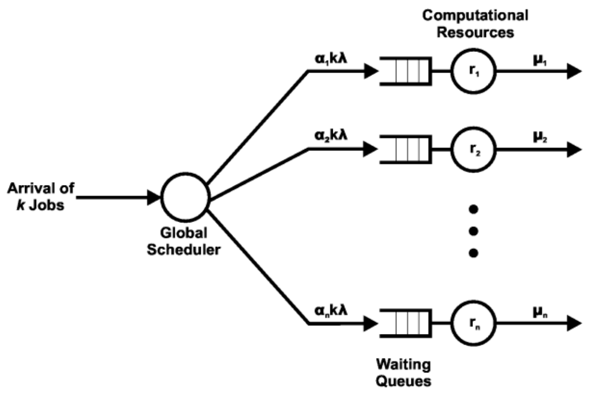

Figure 1. The system model

[3]The White Rose Grid is a multi-site computational grid with heterogeneous compute resources situated at the universities of Sheffield, Leeds and York (see section VI-B for more details).

[4] Each of the computational resources in this system model represents a grid node (i.e. a homogeneous compute cluster).

IV. WORKLOAD ALLOCATION SCHEMES FOR BULK ARRIVALS

a. Weighted Workload Allocation

A simple workload allocation strategy is to set the fraction of workload allocated to each resource to be proportional to the relative processing speed of that resource, i.e.

$$\alpha_i = \frac{\mu_i}{\sum_{j=1}^{n} \mu_j} \qquad (2)$$

This workload allocation strategy takes into account the differences in speeds between resources and attempts to share the workload fairly amongst the resources available. However, [8] have shown that this strategy is suboptimal under most system loads. This weighted workload (WW) allocation algorithm is frequently used in the literature as a benchmark for comparison against static workload allocation algorithms, since it should represent the worst case performance of a static workload allocation algorithm [7], [10].

b. Optimal Workload Allocation for Bulk Arrivals

The problem of allocating workload optimally amongst the resources in a system can be expressed as a non-linear optimisation problem using queueing theory, and thus solved mathematically. The goal of the workload allocation strategy discussed here is to minimise the mean response time of jobs in the system[5]. Each resource in Fig. 1 is modelled as an *M/M/1* queue, with the mean arrival rate of groups of *k* jobs into the system given as λ and the service rate of each resource, r_i, given as μ_i.

According to Little's law [11]:

$$\overline{T} = \frac{\overline{L}}{k\lambda} \qquad (3)$$

where $\overline{T}$ = the mean response time of jobs
$\overline{L}$ = the mean number of jobs in a resource
$k\lambda$ = the mean arrival rate of jobs at a resource

Using queueing theory, the expression for the mean number of jobs in a resource, $\overline{L}$ is found to be:

$$\overline{L} = \frac{k\lambda(k+1)}{2\mu - 2k\lambda} \qquad (4)$$

Therefore,

$$\overline{T} = \frac{k+1}{2\mu - 2k\lambda} \qquad (5)$$

[5] The response time of a job is defined as the time the job spends waiting in the queue plus the time it spends in service.

The mean response time of resource *i* $(1 \le i \le n)$, $\overline{T_i}$, when allocated a fraction of the *k* jobs arriving in each batch, $\alpha_i k_i$ is therefore:

$$\overline{T}_i = \frac{\alpha_i k + 1}{2\mu_i - 2\alpha_i k\lambda} \quad (6)$$

And the mean response time of the system, $\overline{T_{sys}}$, is:

$$\overline{T}_{sys} = \sum_{i=1}^{n} \alpha_i \cdot \frac{\alpha_i k + 1}{2\mu_i - 2\alpha_i k\lambda} \quad (7)$$

The objective of the optimal workload allocation strategy for bulk arrivals is to find the workload allocation, $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, ..., \alpha_n\}$ that minimises $\overline{T_{sys}}$ (and thus optimises the performance of the system), subject to the constraints[6]:

$$\textstyle\sum_{i=1}^{n} \alpha_i = 1 \quad \text{and} \quad \alpha_i k\lambda < \mu_i \quad (8)$$

Rearranging (7) gives:

$$\overline{T}_{sys} = -\frac{1}{2\lambda} \cdot \sum_{i=1}^{n} \left[\alpha_i - \frac{2\lambda\alpha_i + 2\mu_i\alpha_i}{2\mu_i - 2\lambda\alpha_i k} \right] \quad (9)$$

Therefore minimising $\overline{T_{sys}}$ reduces to minimising:

$$F(\boldsymbol{\alpha}) = \sum_{i=1}^{n} \left[\alpha_i - \frac{2\lambda\alpha_i + 2\mu_i\alpha_i}{2\mu_i - 2\lambda\alpha_i k} \right] \quad (10)$$

subject to the constraints in (8).

This is a constrained minimisation problem and can be solved using Lagrange multipliers (see [12] for details), giving:

$$\alpha_i = \frac{1}{2\lambda k} \left[2\mu_i - \sqrt{4\mu_i\lambda + 4\mu_i^2} \cdot \frac{\sum_{j=1}^{n} 2\mu_j - 2\lambda k}{\sum_{j=1}^{n} \sqrt{4\mu_j\lambda + 4\mu_j^2}} \right] \quad (11)$$

If the service rate of a resource is low, it is possible that the workload fraction, α_i, found by (11) may be negative. In this case, the workload allocation, $\boldsymbol{\alpha} = \{\alpha_1, \alpha_2, ..., \alpha_n\}$, produced by (11) will not be practical, since it is not possible to assign negative workload to a resource. To overcome this problem, α_i (where r_i is

[6] The last constraint in (8) ensures that no resource becomes saturated.

the resource with low service rate) should be set to zero, and the workload allocation recomputed without r_i. The pseudocode for calculating the optimal workload allocation for a given set of resources is shown in Fig. 2.

```
procedure WORKLOAD_ALLOCATION($k$, $\lambda$, $\boldsymbol{\mu}$)
    Sort $\boldsymbol{\mu}$ in ascending order
    $m = 1$
    for $i = 1, n$ do
        $\alpha_i = \frac{1}{2\lambda k}\left[2\mu_i - \sqrt{4\mu_i\lambda + 4\mu_i^2} \cdot \frac{\sum_{j=m}^{n} 2\mu_j - 2\lambda k}{\sum_{j=m}^{n} \sqrt{4\mu_j\lambda + 4\mu_j^2}}\right]$
        if $\alpha_i < 0$ then
            $\alpha_i = 0$
            $m = m + 1$
        end if
    end for
    Unsort $\boldsymbol{\alpha}$
end procedure
```

Figure 2. Pseudocode for the workload allocation algorithm

V. PERFORMANCE EVALUATION OF WORKLOAD ALLOCATION ALGORITHMS

a. Experimental Set-Up

Initially the workload allocation algorithms proposed in Section 0 were evaluated using discrete event simulation to investigate their static performance under a variety of system configurations and loads. The simulation model consists of multiple compute resources connected by a high-speed network (see Fig. 1), with data and program files stored on dedicated file servers and thus incurring no appreciable communication overhead from file transfer. Incoming jobs arrive at the global scheduler which then distributes them amongst the compute resources according to the workload allocation calculated by the algorithm. The simulation parameters used in this study have been chosen to represent a realistic system.

Each simulation pass was run from the empty system using a stream of 150,000 job batches. The first 50,000 batches are considered the initialisation period so as to allow the system to reach steady state, and are thus discarded. Statistics about the system are therefore collected from the 100,000 batches of jobs that arrive after this initialisation period, and each point in the performance graphs represents the median of 10 independent runs initialised with different random number seeds.

Previous work [7], [13] has shown that job arrival rates in computer systems are often "bursty" which may cause performance deterioration. One measure of this burstiness is the coefficient of variation (CV) of the arrival process - for example, a Poisson process (as assumed in the theoretical derivation in section 0-B) will have a CV of 1, indicating jobs arrive at regular intervals. Zhou has shown that, for the real-world computing system considered in [13], the trace data collected for the inter-arrival times of jobs exhibits a CV of 2.64 indicating that the instantaneous load on the system fluctuates considerably. Whilst some previous work [7] has modelled the job inter-arrival rate as a two stage hyper-exponential distribution with CV equal to 3, other authors [10] have shown that the performance of both static and dynamic workload allocation algorithms is relatively consistent, despite the higher burstiness of the job arrivals. In the simulation experiments presented in sections V-B, V-C and V-D we will therefore use a Poisson arrival process (as per the theoretical distribution in section VI). However, we will also investigate the impact of ``bursty" job arrival rates on our proposed workload allocation algorithm in section V-E.

The performance of the Dynamic Least Load (DLL) workload allocation algorithm [7] is also included in the following experiments as it represents an upper bound for the static performance of the proposed algorithms. When a job arrives at the global scheduler the DLL algorithm assigns it to the resource with the lowest instantaneous normalised load, thus ensuring that the resources are loaded as lightly as possible. This algorithm requires the global scheduler to keep track of the load index of each resource.

b. Effects of Heterogeneity

Heterogeneity in a computational grid can be measured by the difference in speed between the fastest and slowest computational resource. The effects of varying this speed differential are shown in Fig. 8. The system here consists of 10 computational resources, of which 9 are low speed and 1 is high speed. The speed of the slower resources is fixed at 1, whilst the speed of the faster resource is varied between 1 and 20 - thus the system ranges from a homogeneous system to a highly heterogeneous one. In this experiment jobs arrive at the global scheduler in batches of 10, with the arrival rate set at 0.1 batches per second.

TABLEI. WORKLOAD ALLOCATION IN HIGHLY HETEROGENEOUS SYSTEMS

Heterogeneity	**Proportion of workload allocated to fastest resource**		
	DLL	***ORT***	***WW***
16:1	0.992	0.888	0.640
17:1	0.996	0.909	0.654
18:1	0.998	0.928	0.667
19:1	0.999	0.946	0.679
20:1	0.999	0.963	0.690

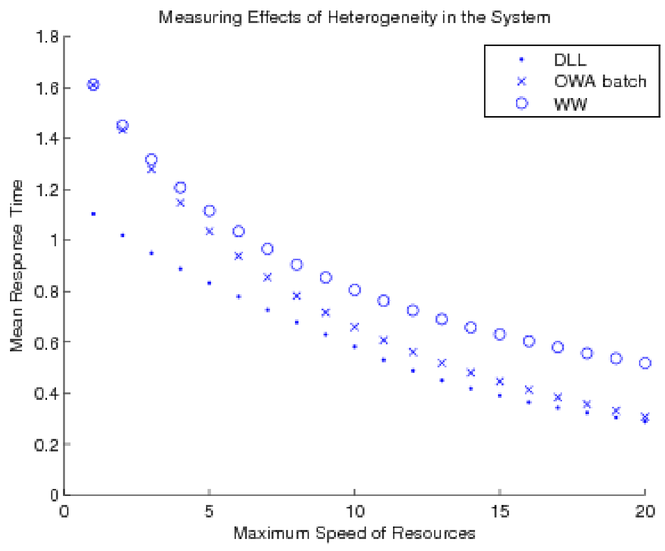

Figure 3. Plot of heterogeneity against performance

It can be seen from Fig. 3 that the optimal workload allocation for bulk arrivals (OWA batch) algorithm significantly outperforms the weighted workload (WW) allocation algorithm in heterogeneous systems. As the heterogeneity of the system increases, so the difference in performance between the OWA batch algorithm and the weighted workload allocation algorithm also increases. When the speed differential between resources is 20:1, the OWA batch algorithm performs 41% better than the weighted workload allocation algorithm. The reason for this (as shown in Table I) is that, in highly heterogeneous systems, the OWA batch algorithm assigns a much larger proportion of the workload to the faster resources than the weighted workload allocation algorithm does. This follows a similar strategy to that used by the Dynamic Least Load algorithm and is the reason why, as the heterogeneity of the system increases, the performance of the OWA batch algorithm approaches that of the Dynamic Least Load algorithm (as can be seen in Fig. 3).

c. Effects of System Size

The scalability of the workload allocation algorithms discussed in this section is found by measuring their performance when the number of computational resources in the system is varied (shown in Fig. 4). In this experiment the system consists of an equal number of low speed and high speed resources. The speed of the slower resources is set to 1, and the speed of the faster resources is set to 10. The number of resources in the system is then varied between 2 and 20. As in the previous experiment, jobs arrive in batches of 10 with the arrival rate set to 0.1 batches per second.

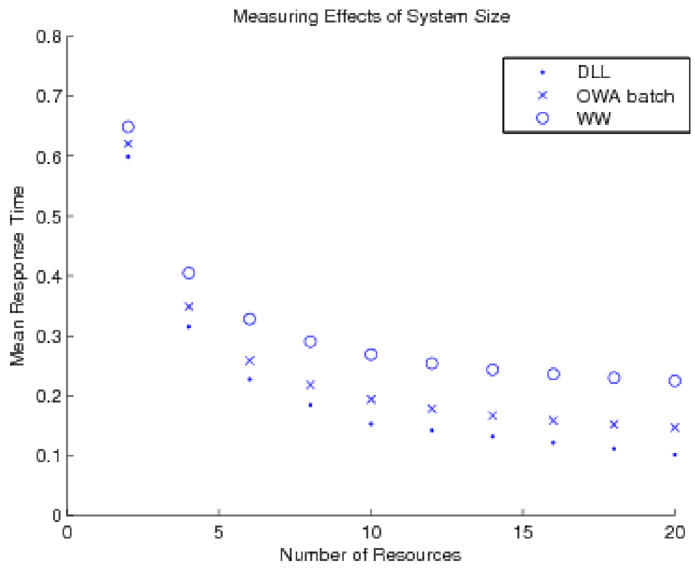

Figure 4. Plot of system size against performance

Fig. 4 shows that the OWA batch workload allocation algorithm performs substantially better than the weighted workload allocation algorithm, and that the difference in performance between the two algorithms increases as the size of the system grows. When there are 6 resources in the system, the OWA batch algorithm outperforms the weighted workload allocation algorithm by 21%. However, when there are 20 resources in the system, this difference in performance increases to 35%. The reason for this difference is that, as explained previously, the OWA batch workload allocation algorithm assigns a disproportionately large fraction of the workload to the fastest resources.

It can likewise be seen from Fig. 4 that the difference in performance between the optimised mean response time algorithm and the Dynamic Least Load algorithm also increases as the system grows. This is because the OWA batch workload allocation algorithm assigns equal fractions of workload to resources that have equal speeds, whilst the Dynamic Least Load algorithm uses instantaneous load information to dynamically balance the workload between resources [7].

Table II shows the difference in the average workload fractions assigned by both algorithms when the system consists of 12 resources. As can be seen in Table II the Dynamic Least Load algorithm assigns different fractions of the workload to resources that have the same speed based on the instantaneous capacity of that resource.

d. Effects of Batch Size

As discussed in section III, jobs arrive at the global scheduler in batches. Fig. 5 shows the impact that the number of jobs arriving in each batch[7] has on the performance of the workload allocation algorithms. The system under consideration in this experiment consists of 10 computational resources with varying speeds (see Table III for the configuration of the system). Batches of jobs arrive at the global scheduler in the system with arrival rate equal to 0.1 batches per second.

TABLEII. WORKLOAD ALLOCATION IN THE 12 RESOURCE SYSTEM FROM SECTION VI-C

Speed of resource	**Proportion of workload allocated to resource**	
	DLL	***ORT***
0	**0**	**0**
10	0.199	0.167
10	0.199	0.167
10	0.198	0.167
10	0.198	0.167
10	0.103	0.167
10	0.102	0.167

It can be seen from Fig. 5 that the optimal workload allocation for bulk arrivals algorithm performs significantly better than the weighted workload allocation algorithm when the number of jobs arriving in each batch is low. However, it can also be seen that, as the batch size increases, the performance of the OWA batch algorithm tends to that of the weighted workload allocation algorithm. When jobs arrive individually at the global scheduler the OWA batch algorithm performs 62% better than the weighted workload allocation algorithm. This difference in performance is reduced to 20% when there are 10 jobs arriving in each batch, and 1.5% when there are 50 jobs in each batch.

[7] The size of the batches of jobs is the primary factor governing the level of workload in the system; the smaller the batch size, the lower the workload level.

TABLEIII. WORKLOAD ALLOCATION IN HIGHLY HETEROGENEOUS SYSTEMS

Speed of resource	Number of resources
1.0	4
2.0	3
5.0	2
10.0	1

The reason for this trend in performance is that the weighted workload allocation algorithm assigns the same fraction of the workload to each resource regardless of the load on the system; whereas, under low system loads, the OWA batch algorithm assigns a disproportionately large fraction of the workload to the faster resources (as explained previously). Since jobs spend less time waiting to be run when the system load is low, assigning a higher proportion of the workload to the faster resources results in smaller response times than if commensurate fractions of the workload were allocated to each resource because the faster resources can process more `work'. However, under higher system loads it is more beneficial to allocate workload in proportion to the speeds of the resources, and thus the OWA batch algorithm behaves more like the weighted workload allocation strategy.

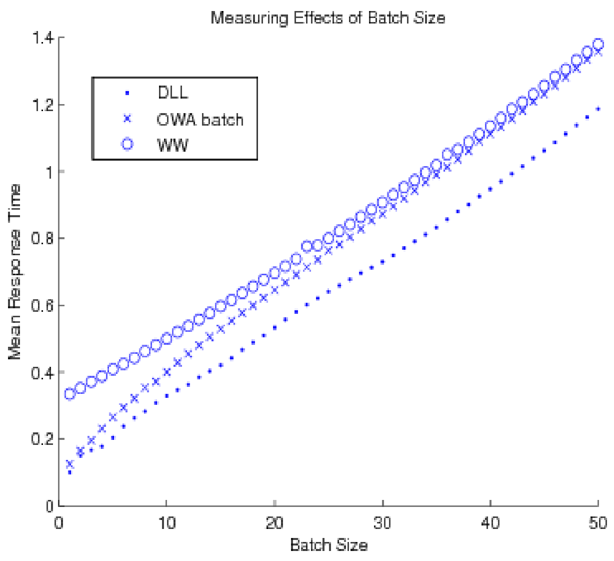

Figure 5. Plot of batch size against performance

Fig. 5 also shows that the difference in performance between the OWA batch algorithm and the Dynamic Least Load algorithm, although small when the system is lightly loaded, increases as the load on the system grows. This is because balancing the load dynamically ensures that the best use is made of the available resources in any given time period. Under heavier system loads this becomes more important because the time that jobs spend waiting for service is more significant. Fig. 6 shows that these performance trends continue when the batch size is very large.

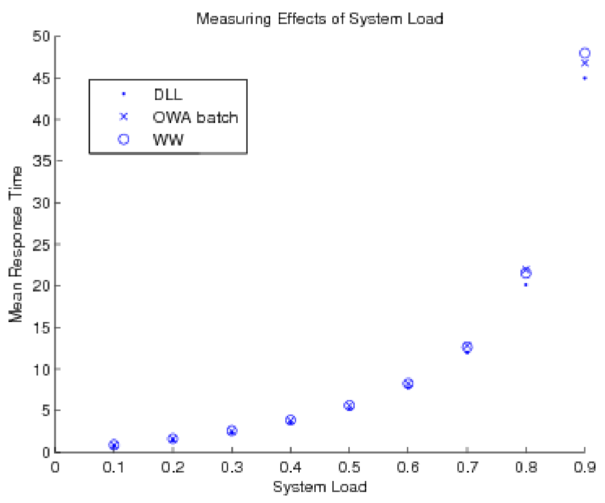

Figure 6. Plot of system load against performance

e. Effects of the Coefficient of Variation

In this section we investigate the impact that ``bursty" job arrivals have on our optimal workload allocation algorithm. Trace data collected from real-world computational grids [15] shows considerable variance in the actual arrival patterns of jobs - with coefficients of variation ranging from close to 1 up to 3. It is therefore important that any workload allocation algorithm performs well across a range of job inter-arrival patterns. In this section a two-stage hyper-exponential distribution is used to model the inter-arrival times of batches of jobs. It can be seen in Fig. 7 that this two-stage hyper-exponential distribution provides a better fit than the exponential distribution to the true job arrival pattern as the CV of the inter-arrival process increases.

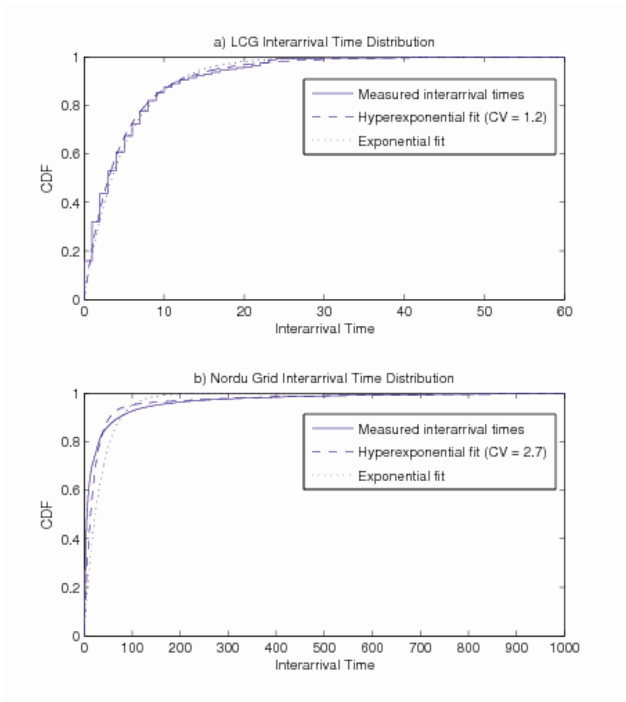

Figure 7. CDF of the inter-arrival times of two real-world computational grids

Fig. 8 shows the effects on the mean response time of jobs through the system of increasing the CV of the job inter-arrival time distribution. In this experiment the system load is fixed at 0.05 and the CV of the job inter-arrival times is increased from 1.0 to 3.0 (simulating arrivals that range from regularly distributed to highly bursty). It can be seen from Fig. 8 that all three of the workload allocation algorithms investigated show an increase in the mean response time of jobs as the CV increases due to the higher instantaneous system loads. However, when the CV of the job arrival process is 3.0 the OWA batch algorithm still outperforms the weighted workload allocation algorithm by 10% and only performs 15% worse than the dynamic least load algorithm. This shows that, even when jobs arrive at the scheduler in an extremely bursty manner, the relative effectiveness of the OWA batch algorithm is not impaired.

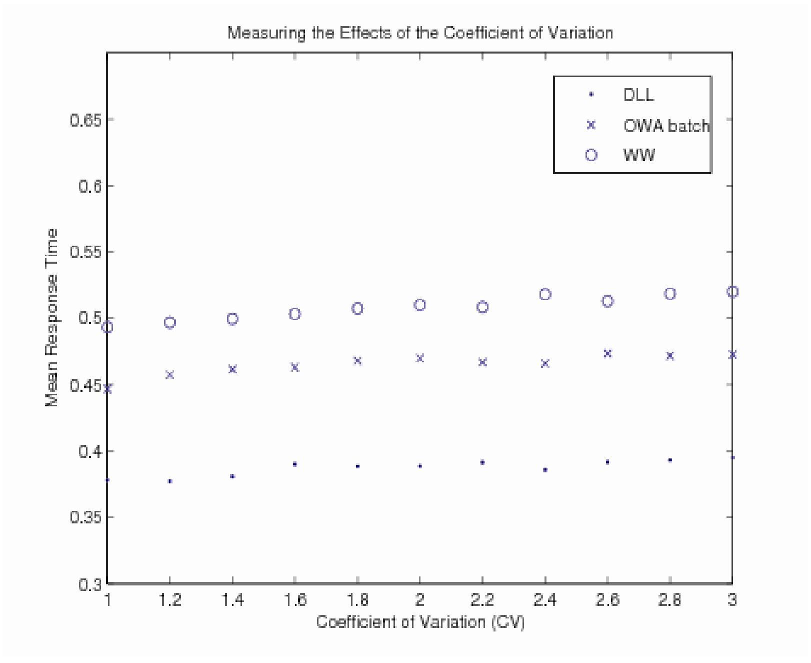

Figure 8. Plot of increasing coefficient of variation against performance

VI. INTRODUCING A NOVEL HYBRID APPROACH TO JOB SCHEDULING

In this section a novel hybrid approach to job scheduling is proposed that combines the low cost of a static scheme with the adaptiveness and efficiency of dynamic strategies. Instead of the difficult and computationally expensive task of collecting frequent detailed information about the state of the system, the proposed hybrid scheduling strategy uses information about the response time of previous jobs through the system to estimate the relative speeds of the compute resources available. It then uses these estimates to schedule the next batch of jobs.

Following the evaluation of the static performance of the proposed workload allocation algorithms using discrete event simulation, a real-world comparison was undertaken between the performance of the hybrid job scheduling approach proposed here and the performance of the static approach used in [7] and [10]. This comparison is described in section VI-B below, and results from a live production grid computing environment are presented.

a. A Hybrid Approach to Job Scheduling

The hybrid approach to job scheduling proposed in this paper was implemented using an application-centric meta-scheduling architecture, with the global scheduler integrated into the application and able to interact with the geographically distributed computational resources via their local resource management systems. As mentioned above, ``best guesses" about the current relative computational speeds of the grid resources were obtained by using historical response time data from recently completed jobs. It then uses the optimal workload allocation scheme for batch jobs proposed in section 0 to calculate the workload allocation for each resource based on this historical data.

What distinguishes the hybrid approach presented in this paper from other proposed hybrid approaches (such as [16] and [17]) is its simplicity and task-centric focus. Whilst [16] and [17] focus on the hybrid scheduling of complex multi-paradigm scientific workflows, the hybrid scheduler proposed in this paper concentrates on the adaptive scheduling of multiple similar tasks that are independent of each other. Whilst this approach is not suitable for tightly-coupled parallel applications such as the n-body problem [18], it has been proven to be effective for many more loosely-coupled applications (such as parameters sweeps and evolutionary optimisation methods [19]).

b. Static Job Scheduling Versus a Hybrid Approach

The low cost hybrid approach to job scheduling proposed in section VI-A above and the static approach widely used in the literature were evaluated in the real-world using the computational resources of the White Rose Grid, a production quality grid computing environment with compute resources located at multiple, geographically distributed sites. The White Rose Grid [20] consists of 4 core nodes - 2 at the University of Leeds, 1 at the University of Sheffield, and 1 at the University of York - providing over 500 processor cores for local users and distributed e-Science research. Each cluster is managed locally using a combination of Sun Grid Engine and the Globus Toolkit and is connected to the rest of the White Rose Grid via the high speed Yorkshire and Humberside Metropolitan Area Network (YHMAN). Fig. 9 shows an overview of the network topology for the White Rose Grid and the geographical layout of these resources.

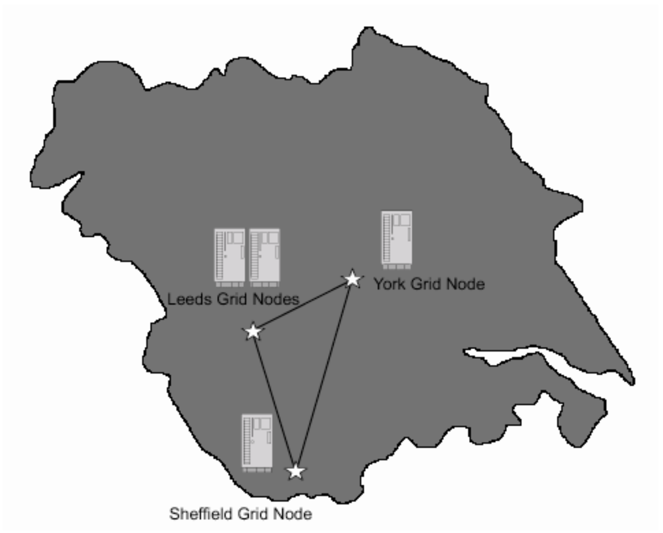

Figure 9. The network topology of the White Rose Grid

Each run of the experiment consisted of both static and hybrid approaches to job scheduling independently allocating 100 sequential batches of jobs amongst the available resources in the grid. The size of each batch was chosen to be 20 jobs, so as to represent a typical real-world application such as a grid search or particle swarm optimisation routine, and 10 independent runs of the experiment were performed at different times of day[8]. The mean arrival rate of batches of jobs at the scheduler and the mean service rates of the grid resources used to calculate the static workload allocation were found by observing the system over a warm-up period immediately prior to the calculation of the workload allocation, whilst the hybrid scheduler was set up so that each resource would be sent the same workload initially (this explains the poor performance for the first iteration of the hybrid job scheduling approach shown in Fig. 10).

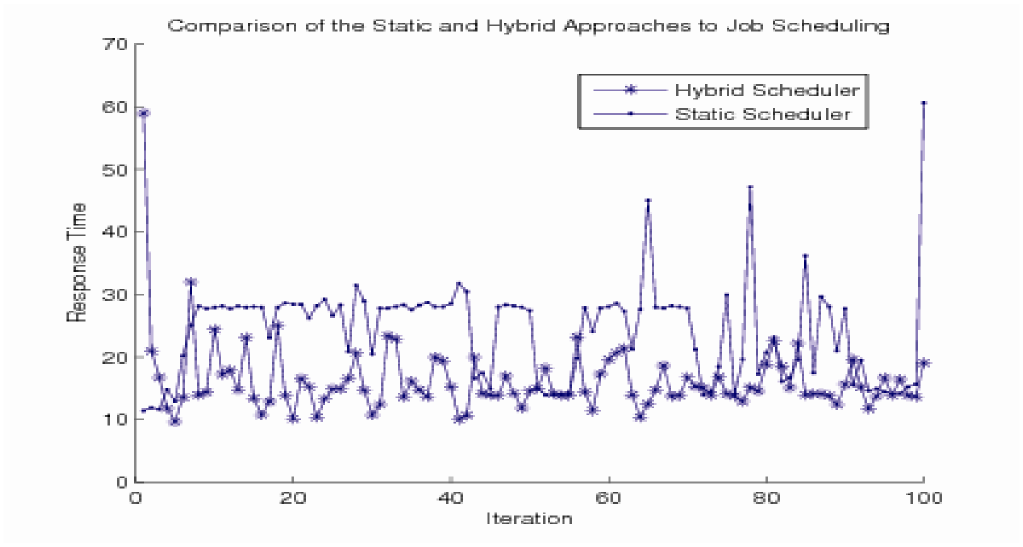

Figure 10. Comparison of the performance of the hybrid and static approaches to job scheduling

Fig. 10 shows that the static scheduling approach initially performs well, with its performance over the first 5 iterations comparable to that of the hybrid approach. However, as the experiment progresses it can be seen that the hybrid approach outlined in this paper significantly outperforms the static scheduler. This is because, whilst the workload allocated using the static approach is optimal with respect to the conditions at the start of the experiment, as those conditions change so the optimal workload allocation also changes. Since the workload allocation is fixed when using the static approach, the workload allocated amongst the grid resources will become sub-optimal as the condition of the system changes[9].

Fig. 11 shows that the hybrid job scheduling approach alters the workload allocation based on the previous response times of the grid resources. In Fig. 12 it can be seen that, in response to the increased response time of the system in iteration 56, the workload allocation for iteration 57 is altered. This increase in response time was due to deterioration in the performance of the grid resource based at the University of York, and therefore the hybrid scheduler reduced the fraction of the workload assigned to that resource and

[8] This was done to investigate the performance of the two approaches under a variety of system loads. The load on the system was typically lowest in the early morning or at the weekend.

[9] For example, if the workload allocation under the conditions at the start of the experiment was {5, 2, 3} and the performance of the first resource deteriorates, then the static scheduler will still assign 5 jobs to that resource - even though the performance of the system may be better if more jobs were assigned to resources 2 and 3.

reallocated it amongst the other available resources in the grid. As can be seen, this results in an improvement in the performance of the system.

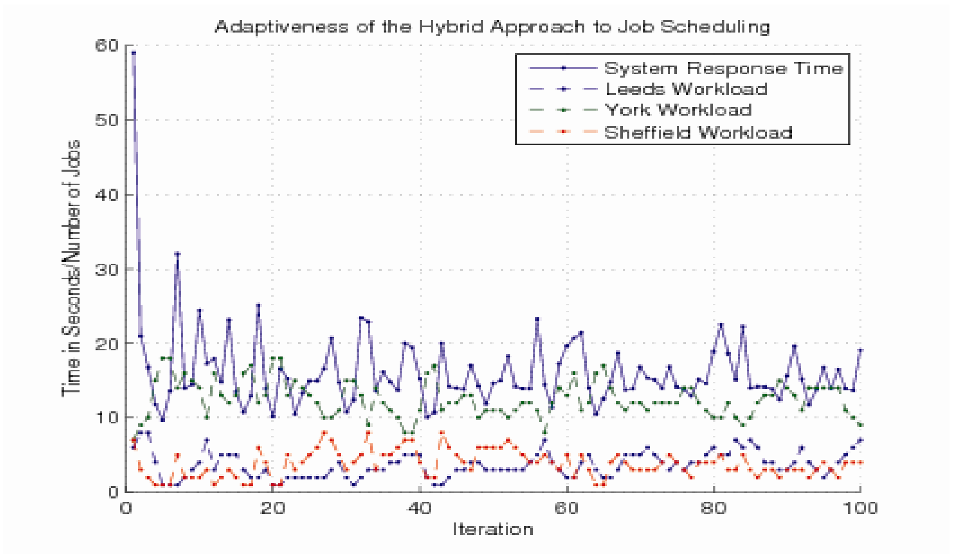

Figure 11. Illustration of the adaptiveness of the hybrid job scheduling approach

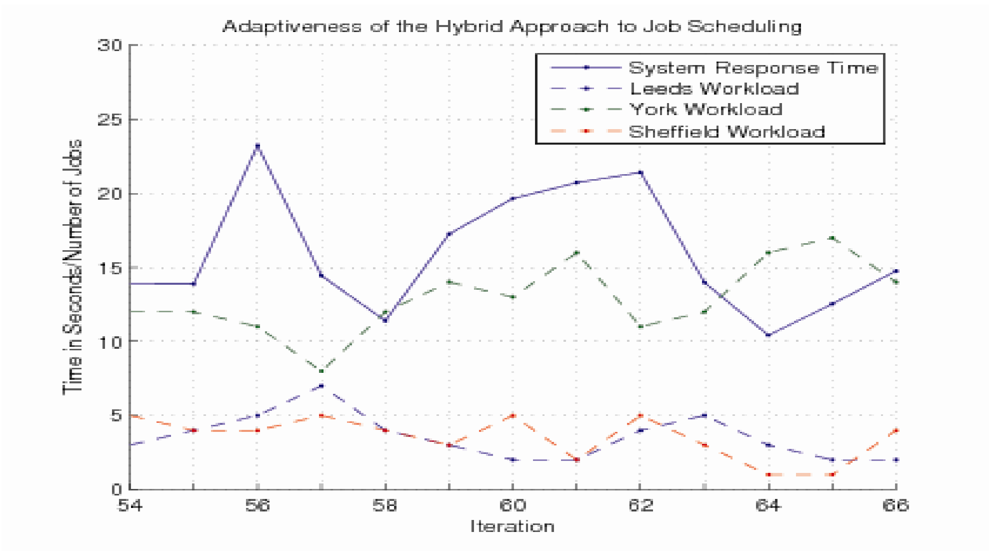

Figure 12. Close-up showing the response of the scheduler to degradation of the performance of one of the resources

The results shown in Figures 10, 11, and 12 are from a single representative experimental run; however, the hybrid approach to job scheduling proposed in this paper performed on average 14.3% better than the static approach over all the runs of the experiment.

VII. CONCLUSION

A key issue in creating production quality grid computing environments is how to distribute sets of computational jobs across the heterogeneous distributed resources that make up a computational grid. This task is complicated by both the decentralised nature of the environment and the complexity of extracting and processing load information from the resources at run-time. Whilst many resource management systems exist to address the problems of scheduling tasks at a local level, it is only quite recently that the idea of a meta-scheduling architecture (whereby a global scheduler sits above multiple local schedulers and interacts with them to submit jobs to the computational resources they control) has been proposed to tackle the key issues of decentralised control and local ownership of resources.

However, whilst meta-scheduling architectures provide a way of tackling issues of job submission and control, workload allocation algorithms and scheduling strategies are needed to decide what proportion of jobs submitted to the global scheduler should be allocated to each heterogeneous distributed resource. Although dynamic approaches frequently outperform static schemes, they usually require the collection and processing of detailed system information at frequent intervals – a task that can be both time consuming and unreliable in complex real-world computational grids [6].

A key contribution of this paper has been the development of a novel workload allocation algorithm to optimise the response time of batches of jobs through the system. Many applications, from parameter sweeps to evolutionary optimisation methods, utilise the processing of multiple independent tasks and can thus benefit from an optimal workload allocation algorithm for batches of jobs. Previous analytical work in deriving optimal workload allocation algorithms has concentrated on jobs arriving individually [7], [10] and these algorithms are unsuitable for the case of multiple jobs arriving together. This paper has also shown that the novel workload allocation algorithm proposed here exhibits substantially better static performance than the weighted workload algorithm explained earlier (a common benchmark in the literature). This performance gap is also consistent regardless of the burstiness of the job inter-arrival process - an important consideration in real-world computational grids where there is considerable variation in the actual arrival patterns of batches of jobs.

Another important contribution of this paper was the development and implementation of a novel hybrid approach to job scheduling that combines the low computational cost of a static scheduling policy with the adaptive capabilities of dynamic methods. This approach uses response time information from previously completed jobs to estimate the current computational capacity of the available resources, and thus make decisions about where to schedule jobs. The novel job scheduling approach proposed in this paper overcomes the need to explicitly query resources for their status (a process that can be both complex and expensive), and instead uses information inherent in the previously completed tasks. As section VI shows, this hybrid approach comprehensively outperforms the same workload allocation algorithm used in the static case.

References

[1] T. L. Casavant and J. G. Kuhl, ``A taxonomy of scheduling in general purpose distributed computing systems," *IEEE Transactions on Software Engineering*, vol. 14, no. 2, pp. 141–154, 1988.

[2] Y. Wang and R. Morris, "Load sharing in distributed systems," *IEEE Transactions on Computers*, vol. 34, no. 3, pp. 204–217, 1985.

[3] L. M. Casey, "Decentralized scheduling," *Australian Computer Journal*, vol. 13, pp. 58–63, 1981.

[4] M. Colajanni, P. S. Yu, and V. Cardellini, "Dynamic load balancing in geographically distributed heterogeneous web servers," in *Proceedings of the 18th International Conference on Distributed Computing Systems*, pp. 295–302, 1998.

[5] S. A. Banawan and N. M. Zeidat, "A comparative study of load sharing in heterogeneous multicomputer systems," in *Proceedings of the 25th Annual Symposium on Simulation*. IEEE Computer Society Press, pp. 22–31, 1992.

[6] J. M. Schopf, "A general architecture for scheduling on the grid," *Argonne National Laboratory, Tech. Rep. ANL/MCS-P1000-1002,* 2002.

[7] X. Tang and S. T. Chanson, "Optimizing static job scheduling in a network of heterogeneous computers," in *Proceedings of the International Conference on Parallel Processing, 2000*, pp. 373–382, 2000.

[8] L. Kleinrock, *Queueing Systems Volume 1: Theory*. New York: John Wiley & Sons, 1975.

[9] R. Leslie and S. McKenzie, "Evaluation of load sharing algorithms for heterogeneous distributed systems," *Computer Communications*, vol. 2, pp. 376–389, 1999.

[10] L. He, S. A. Jarvis, D. P. Spooner, H. Jiang, D. N. Dillenberger, and G. R. Nudd, "Allocating non-real-time and soft real-time jobs in multiclusters," *IEEE Transactions on Parallel and Distributed Systems*, vol. 17, no. 2, pp. 99–112, 2006.

[11] J. D. C. Little, "A proof for the queueing formula: $L = \lambda w$," *Operations*

Research, vol. 9, no. 3, pp. 383–387, 1961.

[12] D. P. Bertsekas, *Constrained Optimisation and Lagrange Multiplier Methods*. Academic Press, 1982.

[13] S. Zhou, "A trace-driven simulation study of dynamic load balancing," *IEEE Transactions on Software Engineering*, vol. 14, no. 9, pp. 1327–1341, 1988.

[14] S. T. Chanson, W. Deng, C.-C. Hui, X. Tang, and M. Y. To, "Multidomain load balancing," in *Proceedings of the 2000 International Conference on Network Protocols (ICNP '00,*, pp. 315–324, 2000.

[15] The Grid Workload Archive, "The grid workloads archive," 2012, viewed 23 March 2012. [Online]. Available: http://gwa.ewi.tudelft.nl/

[16] Z. Yu and W. Shi, "An adaptive rescheduling strategy for grid workflow applications," in *IEEE International Parallel and Distributed Processing Symposium (IPDPS) 2007*, pp. 1–8, 2007.

[17] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: a framework for mapping complex scientific workflows onto distributed systems," *Scientific Programming*, vol. 13, pp. 219–237, 2005.

[18] D. Groen, S. P. Zwart, S. McMillan, and J. Makino, "Pegasus: a framework for mapping complex scientific workflows onto distributed systems," *New Astronomy*, vol. 13, no. 5, pp. 348–358, 2008.

[19] A. Shenfield, P. J. Fleming, V. Kadirkamanathan, and J. Allan, "Optimisation of maintenance scheduling strategies on the grid," *Annals of Operations Research*, vol. 180, no. 1, pp. 213–231, 2010.

[20] The White Rose University Consortium, "The white rose grid website," 2012, viewed 20 April 2012. [Online]. Available: http://www.wrgrid.org.uk/