Volume 67 Number 6 June 2007 ISSN 0743-7315
ELSEVIER
Journal of Parallel and Distributed Computing
EDITORS
Allan Gottlieb, New York University
Kai Hwang, University of Southern California
Sartaj Sahni, University of Florida
• Theory of Parallel/Distributed Computing
• Innovative Computer Architectures
• Supercomputers and Numerical Methods
• Software Tools and Environments
• Parallel Algorithms and Implementations
• Languages, Compilers, and Operating Systems
• Shared Memory Multiprocessors
• Multicomputers and Distributed Processing
• Fault-Tolerant Computing
• Artificial Intelligence Machines
• Applications and Performance Analysis
• Optical Computing
• Neural Networks
Available online at
ScienceDirect
www.sciencedirect.com

Available online at www.sciencedirect.com
ScienceDirect
J. Parallel Distrib. Comput. 67 (2007) 659–673
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

A multi-dimensional scheduling scheme in a Grid computing environment

B.T. Benjamin Khoo[a], Bharadwaj Veeravalli[a,*], Terence Hung[b], C.W. Simon See[c]

[a]Computer Networks and Distributed Systems (CNDS) Laboratory, Department of Electrical and Computer Engineering, National University of Singapore, 4 Engineering Drive 3, Singapore 117576, Singapore
[b]Institute of High-Performance Computing (IHPC), Software and Computing Division, Singapore 117528, Singapore
[c]Asia Pacific Science and Technology Center, Sun Microsystems, Singapore 639798, Singapore

Received 1 February 2006; received in revised form 30 November 2006; accepted 31 January 2007
Available online 20 February 2007

Abstract

In this paper, we propose a novel distributed resource-scheduling algorithm capable of handling multiple resource requirements for jobs that arrive in a Grid computing environment. In our proposed algorithm, referred to as multiple resource scheduling (MRS) algorithm, we take into account both the site capabilities and the resource requirements of jobs. The main objective of the algorithm is to obtain a *minimal execution schedule* through efficient management of available Grid resources. We first propose a model in which the job and site resource characteristics can be captured together and used in the scheduling algorithm. To do so, we introduce the concept of a *n*-dimensional virtual map and resource potential. Based on the proposed model, we conduct rigorous simulation experiments with real-life workload traces reported in the literature to quantify the performance. We compare our strategy with most of the commonly used algorithms in place on performance metrics such as job wait times, queue completion times, and average resource utilization. Our combined consideration of job and resource characteristics is shown to render high-performance with respect to above-mentioned metrics in the environment. Our study also reveals the fact that MRS scheme has a capability to adapt to both serial and parallel job requirements, especially when job fragmentation occurs. Our experimental results clearly show that MRS outperforms other strategies and we highlight the impact and importance of our strategy.
© 2007 Elsevier Inc. All rights reserved.

Keywords: Grid computing; Scheduling; Parallel processing time; Multiple resources; Load distribution

1. Introduction

With recent technological advances in computing, the cost of computing has greatly decreased, bringing powerful and cheap computing power into the hands of more individuals in the form of commodity-off-the-shelf (COTS) desktops and servers. Together with the increasing number of high bandwidth networks provided at a lowered cost, use of these distributed resources as a powerful computation platform has increased. Vendors such as the IBM [24,9], the HP [8] and the Sun Microsystems [27] have all introduced clusters that would effectively lower the cost-per-gigaflop of processing while maintaining high performance using locally distributed systems. The concept of Grid computing [5] has further pushed the envelope of distributed computing, moving traditionally local resources such as memory, disk and CPUs to a wide area distributed computing platform sharing these very same resources. Consequently, what had used to be optimal in performance for a local cluster has suddenly become a serious problem when high latency networks, uneven resource distributions, and low node reliability guarantees, are added into the system. Scheduling strategies for these distributed systems are also affected as more resources and requirements have to be addressed in a Grid system. The lack of centralized control in Grids has also resulted in failure of traditional scheduling algorithms where different policies might hinder the sharing of specific resources. This leads to a lack of robust scheduling algorithms that are available for Grids.

In this paper, we propose a novel scheme that considers various resource requirements of jobs while taking into consideration the distributed computation environment where the job resides in. The technique we propose shall then devise an allocation, which can be used to provide what it believes as

* Corresponding author. Fax: +65 67791103.
E-mail address: elebv@nus.edu.sg (B. Veeravalli).

0743-7315/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2007.01.008

the most efficient job execution sequence to handle the jobs. Below we summarize our contributions in this paper.

1.1. Our contributions

We propose a novel methodology referred to as multiple resource scheduling (MRS) strategy that would enable jobs with multiple resource requirements to be run effectively on a Grid computing environment (GCE). A job's resource dependencies in computational, data requirements and communication overheads will be considered. A parameter called Resource Potential is also introduced to ease in situations where in inter-resource communication relations need to be addressed. An *n*-dimensional resource aggregation and allocation mechanism is also proposed. The resource aggregation index, derived from the *n*-dimensional resource aggregation method, and the Resource Potential sufficiently allows us to mathematically describe the relationship of resources that affects general job executions in a specific dimension into a single index. Each dimension is then put together to form an *n*-dimensional map that allows us to identify the best allocation of resources for the job. The number of dimensions considered depends on the number of job related attributes we wish to schedule for.

The combination of these two methodologies allows MRS to be able to respond more suitably in the execution of applications that are both highly parallel as well as serial in nature in GCEs. The performance of such a scheduling algorithm promises respectable waiting times, response times, as well as an improved level of utilization across the entire GCE.

As dimensional indices are computed at the resource sites itself, this vastly improves the distributed control of the Grid over resources. It additionally unloads scheduling overheads due to resource comparison at the main scheduling server. This design also paves way in designing a distributed scheduling system as each additional resource is responsible for its own sharing of resources and computation of indexes. This naturally allows the MRS to be possibly implemented easily as both a central and distributed scheduling systems. In this paper, we restrict the scope of simulation to a central scheduling design of the MRS. However, we will present a discussion how a distributed MRS system can be easily achieved.

We evaluate the performance of our proposed strategy in two dimensions, namely computation and data, while addressing requirements of resources such as, FLOPS, RAM, Disk space, and data. We study our strategy with respect to several influencing factors that quantify the performance. Our study shows that MRS outperforms most of the commonly available schemes in place for a GCE.

1.2. Organization of paper

The organization of this paper is as follows. In Section 2, we describe the GCE and in Section 3 we introduce our MRS strategy and algorithm. Section 4 describes the system design and implementation of the system. We also present our rigorous performance evaluation and discuss the significance of the results in this section. In Section 5, we introduce the relevant literature in this domain and highlight some key works in this area. Section 6 concludes the paper.

2. Grid environment model

In this section, we define the GCE in which the MRS strategy was designed. We first clearly identify certain key characteristics of resources as well as the nature of jobs. A GCE comprises many diverse machine types, disks/storage, and networks. In our resource environment, we consider the following:

1. Resources can be made up of individual desktops, servers, clusters or large multi-processor systems. They can provide varying amounts of CPU FLOPs, RAM, Harddisk space and bandwidth. Communication to individual nodes in the cluster will be done through a local resource manager (LRM) such as SGE, PBS, or LSF. We assume that the LRM will dispatch a job immediately when instructed by the Grid meta-scheduler (GMS). The GMS thus treats all resources exposed under a single LRM as a single resource. We find this assumption to be reasonable as GMS usually does not have the ability to directly contact resources controlled by the LRM.
2. Changes in any shared resource at a site is known instantaneously to all locations throughout the GCE. Without loss of generality we assume that every node in the GCE is able to execute all jobs when evaluating the performance of the MRS strategy.
3. Each computation resource is connected to each other through different bandwidths which are possibly asymmetrical.
4. All resources have prior agreement to participate on the Grid. From this, we safely assume a trusted environment whereby all resources shared by sites are accessible by every other participating node in the Grid if required to do so.
5. In our simulations, we assume that the importance of the resources with respect to each other is identical.
6. The capacity for computation in a CPU resource is provided in the form of GFlops. While we are aware that this is not completely representative of a processor's computational capabilities, it is at current one of the most basic measure of performance on a CPU. Therefore, this is used as a gauge to standardize the performance of different CPU architectures in different sites. However, the actual units used in the MRS strategy does not require actual performance measures, rather, it depends on relative measures to the job requirements. We will show how it is done in later sections.

The creation of the job environment is done through the investigation of the workload models available in the Parallel Workload Archive Models [18] and the Grid workload model available in [25]. The job characteristics are thus defined by the set of parameters available in these models and complemented with additional resource requirements that are not otherwise available in these two models. Examples of these resources include information such as job submission locations and data size required for successful execution of the task. In our job execution environment, we assume the following:

1. Resource requirement for a job does not change during execution and are only of (a) single CPU types, or (b) massively parallel types written in either MPI such as MPICH [1] or PVM. [2]
2. The job resource estimates provided are the upper bound of the resource usage of a given job.
3. Every job submitted can have its data source located anywhere within the GCE.
4. A job submitted can be scheduled for execution anywhere within the GCE. Without loss of generality, we also assume that the applications to be executed are already available in all sites within the GCE.
5. Job's resource requirements are divisible into any size prior to execution.
6. In addition to computational requirements (i.e., GFlops, RAM and File system requirements), every job also has a data requirement whereby the main data source and size is stated. These data resources required are accessible using GridFTP or GASS [3] services provided by the Globus Toolkit.
7. The effective run time of a job is computed from the time the job is submitted, till the end of its result file stage-out procedure. This includes the time required for the data to be staged in for execution and the time taken for inter-process communication of parallel applications.
8. Resources are locked for a job execution once the distribution of resources start and will be reclaimed after use.

A physical illustration of the resource environment that we consider is shown in Fig. 1, and the resource view of how the GMS will access all resources through the LRM is shown in Fig. 2.

3. Scheduling strategy

From Section 2, we note that the system environment of the Grid consists of heterogeneous nodes (both desktops as well as servers). This results in an environment whereby a wide range of resources are available. These resources may or may not be well connected to each other depending on network connectivity and thus require proper allocation and grouping before jobs can be executed efficiently.

The MRS strategy considers job requirements and resource capabilities based on some performance metrics and compute the best matching sites for a job to be dispatched to. It also encodes common inter-resource dependencies that affects the efficient execution of jobs, including I/O dependence and communication overheads in its decision making process. This allows jobs to be executed efficiently when allocated to resources that span across many remote sites. We term this phenomenon as "job fragmentation". This occurs when a single serial or parallel job is allocated to a site that is unable to, by itself, satisfy all the resource requirements needed. It thus needs to obtain additional resources from remote sites in order to execute successfully. MRS is able to allocate both serial and parallel jobs (optimally) as it always prefers allocating to sites that are of high bandwidth and low communication overheads. MRS also treats each submitted job as an independent entity and does not address workflow requirements of any application. We feel that this is done without any loss of generality as workflow requirements should be addressed at an orchestration layer independent of the scheduling middleware. With MRS always allocating every job to sites that best provides its resources, it ensures that the job execution environment will always be the best for both serial as well as parallel jobs.

Without loss of generality, jobs request and site representations of CPU resources is done in terms of GFLOPs as an indication of performance. Future changes in unit representations will not affect the strategy as the aggregation algorithm will result in dimensionless indexes as long as the request and site resource representation units are the same. This applies to all other resources shared within MRS.

MRS also tries to allocate resources such as to satisfy a job's requirements in a single site in order to improve performance. It additionally avoids over allocation of resources, so as to prevent the detrimental effects on other jobs which might need these resources to achieve efficiency in execution. By mapping a job to the best capable and matched resources, we hope to be able to improve the following metrics of performance measure.

1. Average wait-time (AWT): This is defined as the time duration for which a job waits in the queue before being executed. The wait time of a single job instance is obtained by taking the difference between the time the job begins execution (e_j) and the time the job is submitted (s_j). This is computed for all jobs in the simulation environment. The average job waiting time is then obtained. If there are a total of J jobs submitted to a GCE, the AWT of a job is given by

$$AWT = \frac{\sum_{j=0}^{J-1}(e_j - s_j)}{J}.$$

   This quantity is a measure of responsiveness of the scheduling mechanism. A low wait time suggests that the algorithm can potentially be used to schedule increasingly interactive applications due to reduced latency before a job begins execution.
2. Queue completion time (QCT): This is defined as the amount of time it takes for the scheduling algorithm to be able to process all the jobs in the queue. This is computed by tracking the time when the first job enters the scheduler until the time the last job exits the scheduler. In our experiments, the number of jobs entering the system is fixed, to make the simulation more trackable. This allows us a quantitative measure of throughput, where the smaller the time value, the better. The QCT is given by

$$QCT = e_{J-1} + E_{J-1} - s_0,$$

[1] MPICH: http://www-unix.mcs.anl.gov/mpi/mpich/.
[2] Parallel Virtual Machines: http://www.csm.ornl.gov/pvm/pvm_home.html.
[3] Grid Access to Secondary Storage: http://www.globus.org/gass.

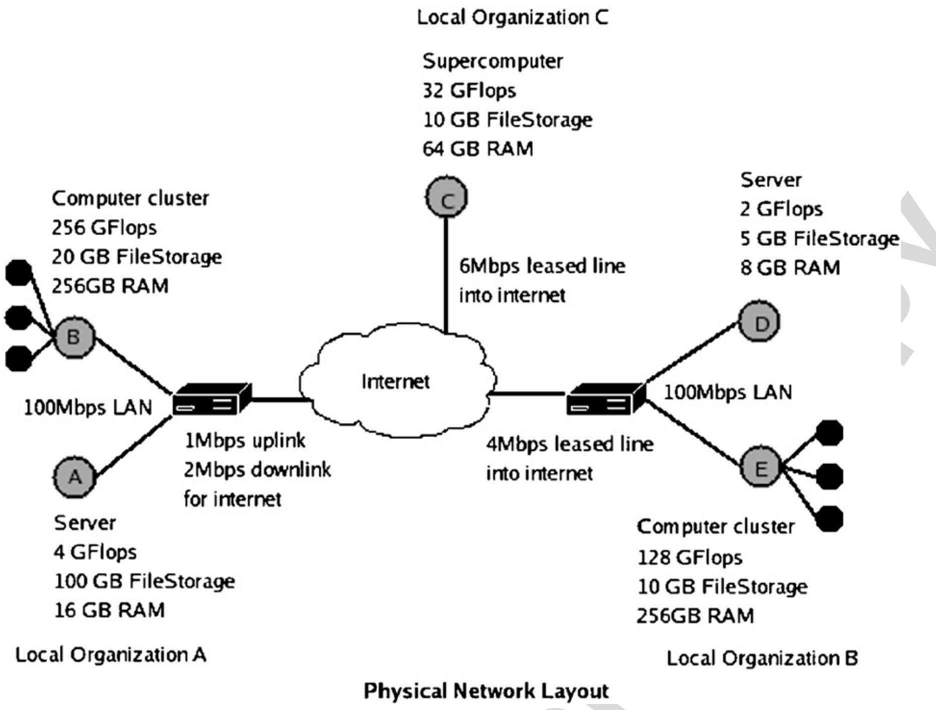

Fig. 1. Illustration of a physical network layout of a GCE.

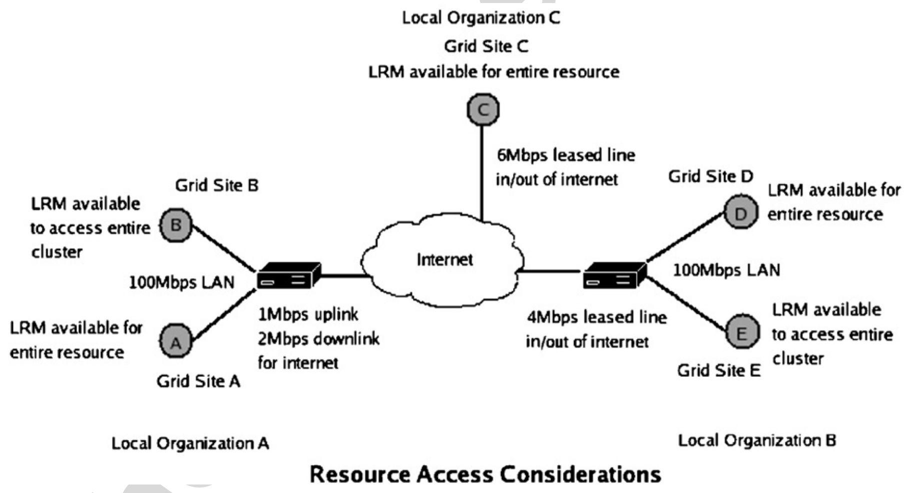
Local Organization C
Grid Site C
LRM available for entire resource
6Mbps leased line in/out of internet
Grid Site B
LRM available to access entire cluster
Grid Site D
LRM available for entire resource
Internet
100Mbps LAN
100Mbps LAN
1Mbps uplink
2Mbps downlink
for internet
4Mbps leased line in/out of internet
LRM available for entire resource
Grid Site A
LRM available to access entire cluster
Grid Site E
Local Organization A
Local Organization B
Resource Access Considerations

Fig. 2. Resource view of physical environment with access considerations.

where E_{J-1} is the execution time of the last job. This includes the I/O and communication overheads that occurs during job execution.
This metric, when coupled with the average waiting time of a job, allows us to deduce the maximum amount of time a typical job will spend in the system for a given workload.

3. Average grid utilization (AGU): This quantity investigates how well the algorithm is capable of organizing the workload and the GCE resources so as to optimize the performance. Thus, the higher the utilization, the better optimized the environment is. The utilization of the GCE at each execution time step is captured and represented as $U(t) = \frac{M_u}{M}$, where M is the total computational resources available. M_u is the number of computational resources utilized. The AGU is thus given by the following equation:

$$AGU = \frac{\sum_{t=s_0}^{QCT} U(t)}{QCT}.$$

3.1. MRS dimension allocation and resource index aggregation

As stated in Section 1.1, MRS is a n-dimensional allocation strategy. In order to make use of this strategy, the dimensions to consider must first be decided. The dimensions should be the general classifications of resource requirements that would be required by a job. We make use of two basic dimensions: (1) Computation, and (2) Data, in our simulations in order to verify the effectiveness of our strategy. These two dimensions are chosen due to the general requirement to achieve faster computation through proper resource allocation such as GFLOPs, RAM and disk, and better data resource allocation to achieve higher I/O throughput. Aggregation of the various available resources are then combined into two major indices based on these two dimensions. We refer to these indices as the Computational and Data Index, respectively.

From the two indices, we create a 2-dimensional (2D) plot with the Computation and Data Index. This 2D plot describes the virtual topology of the job resource requirements, situated at the origin, to the resource providing sites in the GCE. We call this virtual topology a *Virtual Map*. It is thus clear that each site has two indices that describes its suitability for the job. The most suited resource providers will be the sites whereby it is located nearest to the origin. The sections below will demonstrate how we construct the two selected dimensions and the process of aggregation that leads to the final aggregated Indexes used in the Virtual Map.

3.2. *Computation dimension*

Resources in the computation dimension consist of entities that would impact the efficient computation of a job. Each resource is in turn represented by a capability value and a requirement value. In our simulations, we make use of the following allocable resources as basis for scheduling in the computation dimension:

- GFLOP (C),
- RAM (M),
- Disk space (F).

However, we note that this is insufficient to represent a collection of sites and how they can possibly inter-operate with each other. A job submitted to a poorly connected site will be penalized when job fragmentation occurs or when the data required for processing is located in another location.

In order to minimize the detrimental effects in such cases, we introduce a parameter referred to as the *Resource Potential*. This is to assist in the evaluation of the Computation Index. We denote m as the total number of sites in a GCE. The potential, denoted as P_i, of a resource R_i quantifies the level of network connectivity between itself and its neighboring sites. For simplicity, we assume that the network latencies as well as the communication overhead of a resource is inversely proportional to its bandwidth. We refer to the Resource Potential, P_i of a resource R_i, as a form of "Virtual Distance", where $1 \leqslant i \leqslant m$. This is computed as $P_i = \sum B_{ij}$ where, B is the upload bandwidth, expressed in bits per sec, from R_i to R_j for $i \neq j$ and $B_{ij} = 0$ if $i = j$. This effectively eliminates all network complexities and "flattens" the bandwidth view of all the resources to the maximum achievable bandwidth between resources. This also inherently includes all sub-net routing overheads and communication overheads when a bandwidth monitoring system such as NWS [32] is employed. We illustrate this "flattening" process in Fig. 3. The values C, M, F and P_i dynamically change with resource availability over time t, and is constantly monitored for changes in our simulation. Thus, in a GCE where we characterize the resource environment as a set $S = \{R_1, \ldots, R_m\}$, we can represent the allocatable computational resources within a site i as a set $S_c = \{R_i, t\}$ where $S_c \subseteq S$. R_i is further represented by 4-tuple of $f_i(\langle C, M, F, P_i\rangle, t)$ denoting the four resources considered in our allocation strategy.

In order to ascertain an aggregated Computation Index of a site to a job, resources are also requested based on the same GFLOPs, RAM and Harddisk space required. Similar to a node's Resource Potential described earlier, jobs are also additionally characterized by a potential value. However, this potential value is not obtained from the location where the job is submitted from, rather, it is obtained from the location of the source file required for the job to execute efficiently. In our simulations, we assume that each job only requires data from one data resource. This data resource can be either local to the job submission site or remote. As MRS is expected to operate in a GCE, we also simulate scenarios wherein users can submit jobs from different locations. [4]

We characterize the job environment by $J = \{A_i, \ldots, A_j\}$, and the computational requirement of each job A_j in the set of J jobs is represented by $g_j(\langle C, M, F, P_{\text{src}}\rangle, t)$.

3.3. *Computational Index through aggregation*

Evaluation of various resource requirements of sites and jobs allows us to aggregate their values and encode inter-resource relationships in order to arrive at a single computational index such that it can be used to obtain the allocation score. This is done by obtaining a ratio of provision (R_{ij}), for site i and job j, between what is requested and what is possibly provided. For computational resources, it is given by, $R_{ij}\{C\} = 1 - \frac{f_i\{C\}}{g_j\{C\}}$. Only the positive values of $R_{ij}\{C\}$ are considered, such that and $R_{ij}\{C\} = 0$ if the above evaluates to be less than zero. $f_i\{C\}$ and $g_j\{C\}$ are the GFLOP resource provided at site i and GFLOP resource required by job j. We only consider positive values in the Virtual Map, and therefore truncate the values at zero. We make several observations in this equation.

1. Perfect ability to provision for a resource results in this value being 0.
2. Inability to provide for a resource results in $0 < \frac{f_i\{C\}}{g_j\{C\}} < 1$. The $R_{ij}\{C\}$ value would approach 1 as the inability to provision a resource to a job increases.
3. Over-ability to provision resources for a job results in the $R_{ij}\{C\} = 0$.

We apply the same ratio of provision to all resource and requirements within the computational dimension which also includes RAM (M) and Harddisk (F) requirements. Additionally we also include the ratio of provision between the

[4] Without loss of generality, we have assumed that applications are pre-staged at the sites.

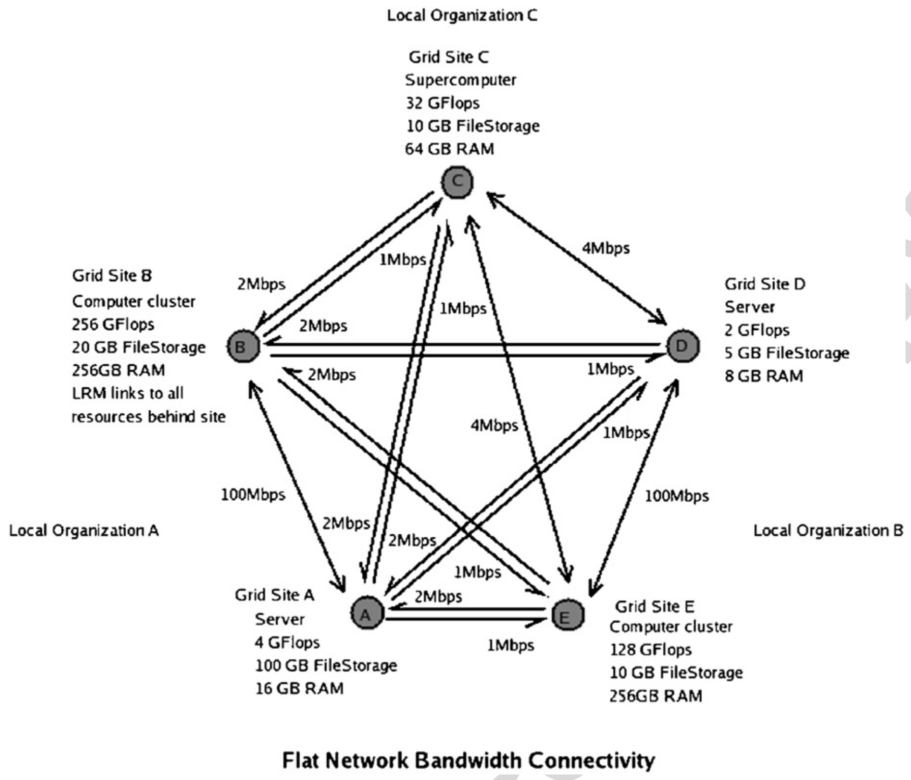

Fig. 3. Flattened network view of resources for computation of potential.

potential value of the site (P_i) and the source file potential (P_{src}). This allows us to evaluate if a site connectivity is equal or better to where the source data file is located. This ensures that the possible target job submission site will not be penalized more than required if job fragmentation is to occur, when compared to executing the job in place at the data source location.

These ratios are then aggregated into a dimensionless computation index (x_i) for site i on job j using the following equation. Constants K_C, K_M, K_F and K_P represents weights that provide modification to the importance of the respective provisioning ratios in terms of importance to each other. An increasing value of $K > 0$ signifies an increasing importance of a specific resource requirement relative to the other resources. This steers the strategy away from the default allocation to one that is weighted towards the more important resource.

After the sites providing resources are indexed to obtain x_{ij}, the site i with the lowest computation index, x^*_{ij} is deemed to provide the best resources suited for a job j. In our simulations, we set the K constants such that $K = 1$,

$$x_{ij} = \sqrt{(K_C R_{ij}\{C\})^2 + (K_M R_{ij}\{M\})^2 + (K_F R_{ij}\{F\})^2 + (K_P R_{ij}\{P\})^2}. \tag{1}$$

Fig. 4 illustrates how the K constants modifies the result of Eq. (1). The figure shows two sites A and B, which both evaluates to the same x_{ij} value if $K = 1$ for an arbitrary job. Site A provides a perfect fit for a job in RAM while Site B provides a perfect fit for the job in disk space. Under normal circumstances, both sites would be equally considered in job allocation. However, Fig. 4 shows that as K_M is decreased to 0, representing the lowered importance of RAM to the other required resources, Site B begins to become a more favorable allocation site in view of its computational index. Site B thus gets a better chance of being selected for allocation even though Site A perfectly fits the job requirements for RAM. This is due to the fact that the allocation mechanism has been steered from considering the effects of RAM, and is now giving more consideration to the other resources such as disk space instead. As K_M increases beyond 1, it can be observed that RAM has now gained greater importance w.r.t. the other resource requirements, resulting in a increasing deviation between the x_{ij} values of Sites A and B. In this circumstance, the strategy would thus select Site A as it fits the RAM requirements perfectly and not select Site B based on the minimum value of x_{ij}.

3.4. Data dimension and indexing through resource inter-relation

In the data dimension, we wish to inter-relate resources that would affect the I/O of a job and evaluate an index that aids us in determining a good resource site that would best execute a job. The expected time for I/O is determined based on the estimated data communications required and the bandwidth between the source file location and the target job allocation site. The ratio between the I/O communication time to the estimated local job runtime is then taken. This ratio allows us to evaluate the level of advantage a job has in dispatching that job to a remote site.

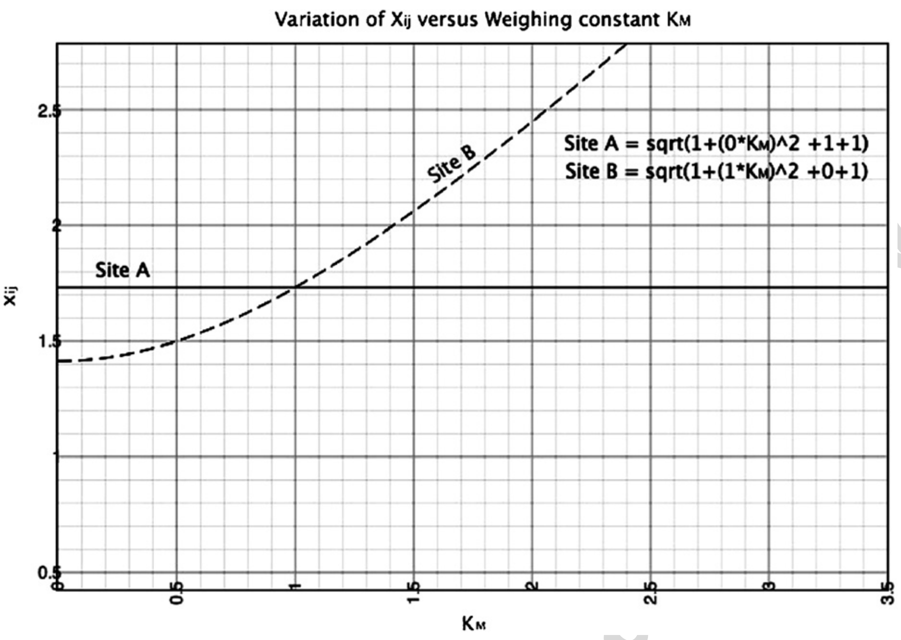

Fig. 4. Variation of x_{ij} versus changes in the weighing constant K_M.

This is because a site capable of executing a job locally would incur a minimal (not-zero) I/O time as compared to any other remote location. Thus, allocation of a job to the intended target resource should be one whereby this ratio is as low as possible.

The I/O time is mainly dependent on the availability of bandwidth at a site. The available bandwidth also changes over time depending on if a resource is sharing any of its network resources with other resources in the GCE. This is also captured as a sequence of complete network allocation for a job in our simulator. We annotate bandwidth B between two sites i and j as $B_{ij} = \min\{B_{ij}^{\text{download}}, B_{ji}^{\text{upload}}\}$ which changes over time t as data capabilities of a resource $S_d\{R_i, t\}$. Where each item in the set is represented by $d_i\{\langle B\rangle, t\}$. The data requirement of a job j is thus represented by $e_j\{\langle F, A^{\text{runtime}}\rangle, t\}$ where A^{runtime} is the estimated runtime of the job.

We make use of this ratio to create the Data Index. This evaluation is an example of aggregation based on resource inter-relation. I/O time is affected by the amount of data for a job and the actual bandwidth resource available. In the worst-case scenario, the amount of data required for the job would also be the amount of harddisk resource required at the site to store the data to be processed. This, therefore inter-relates the data resources to the bandwidth resources available. The ratio is written as follows:

$$y_{ij} = \frac{e_j\{F\}}{d_i\{B_{ij}\}} \cdot \frac{1}{A^{\text{runtime}}}. \tag{2}$$

It is noted that y_{ij} continues to be dimensionless and a smaller value would represent a better site i preference when compared to a larger one. An (ascending) ordered y_{ij} would rank sites with the better advantage in handling job fragmentation compared to those ranked later.

3.5. Dimension merging

From the individual Computation and Data Indices described above, we observe that the best allocated resources are represented by those with low index values. Each of the individual indices are also encoded with resource requirements considerations in its evaluation through aggregation. These points when plotted on a 2D axis creates what we termed as the Virtual Map that is described in Section 3.1. As we have observed, sites that position themselves closest to the origin are those that deviate from the resource requirements by the least amount. An illustration of the Virtual Map is shown in Fig. 5. The Euclidean distance from the origin therefore denotes the best possible resources that matches the resource requirements of a job for an instance in time.

In Fig. 5, the computation and data index is computed by Eqs. (1) and (2) for each job in the queue. As job requirements differs for each job, the Virtual Map is essentially different for each job submitted. This has to be computed each time a job is to be submitted or re-submitted to the GCE.

4. System design and implementation

From the allocation strategy, we proceed to implement the scheduling mechanism in a GCE. There are several points observed in the implementation of the system made to support the MRS scheduling strategy.

- Each dimensional index is independent between sites and can therefore be computed locally at the participating sites within the GCE.
- A job can be submitted from any node within the GCE. A resource requirement broadcast mechanism with timeout was implemented for each job. This allows the job to announce

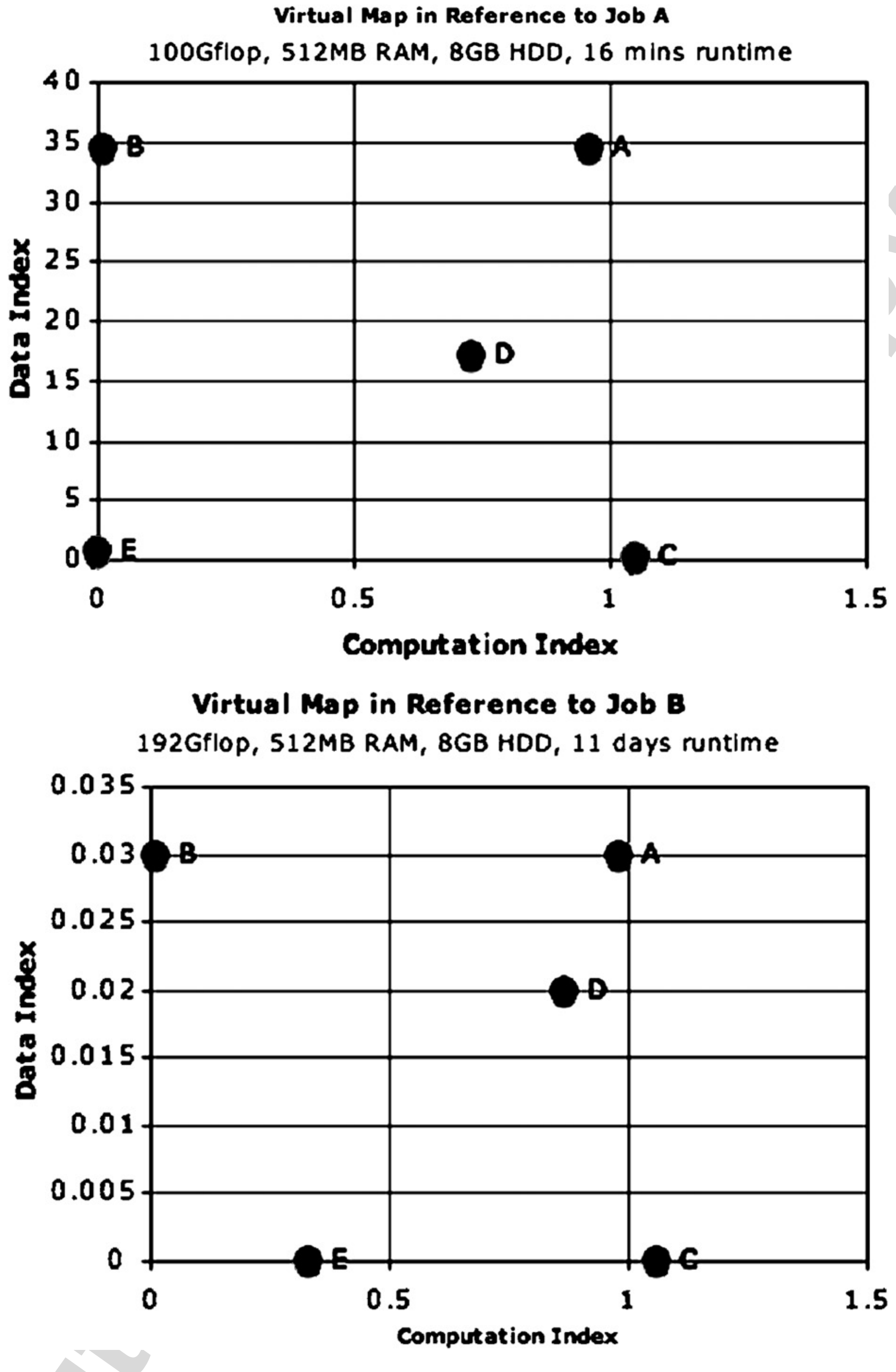

Fig. 5. A Virtual Map is created for each job to determine allocation.

itself to the sites within the GCE. It also allows sites within the GCE to obtain the specific requirements for each job and evaluate its computation and data indexes accordingly. The timeout for requirements broadcast effectively truncates sites that do not reply within a certain delay. This is a simple mechanism to efficiently truncate sites that are responding too slowly to requests due to high load or congested bandwidth.

- A caching mechanism was implemented in the participating sites in order to help reduce the communications overheads. As resources are not always available to handle jobs at the time of job submission, these job requirements would have to be resent in a broadcast whenever a change in resource availability is detected.

A job allocation communication in the event of a new job submission is as follows.

1. New job announces itself to entire GCE using a unique job ID together with its requirements.

2. Sites receive broadcast and acknowledge (ACK) with Euclidean distance of its Virtual Map location to origin. Sites also cache the requirements locally.
3. Job submission location waits for a timeout and collects all ACK responses and sorts the results in ascending order.
4. Full job description is dispatched to the target site.
5. Target site acknowledges receipt of full job and begins processing execution request. Job submission location then issues a "cache clear" to all sites for this job ID.

In event of a job re-submission, the following process takes place first for a job with the longest wait time.

1. Job re-announces submission request to the GCE.
2. Sites with requirements in cache ACK with Euclidean measure. Sites without requirements in cache ACKs with requirements request.
3. Job submission location sends requirements to the other sites and waits for timeout.
4. Sites receive requirements and acknowledge (ACK) with Euclidean distance of its Virtual Map location to origin. Sites also cache the requirements locally.
5. Job submission location waits for a timeout and compiles all ACK responses and sorts the results in ascending order.
6. Full job description is dispatched to the target site.
7. Target site acknowledges receipt of full job and begins processing execution request. Job submission location then issues a "cache clear" to all sites for this job ID.

From the above process, several advantages in the implementation of the MRS strategy is observed.

- The indexes, being independent between sites, are evaluated within sites. It does not require any system monitoring mechanism to inform a master scheduler about its state. Thus reducing the complexity of the entire system during implementation.
- The main scheduler in the MRS does not contain complex algorithms and is only required to sort the resulting Euclidean measure that is obtained from the GCE. Only job tracking functionalities are required at the various locations where job submission is permitted. This allows the strategy to scale better when more resources are added into the GCE to be considered. It also allows more inter-resource relations to be defined as separate dimensions without computational penalties as in a central scheduling strategy.
- Multiple MRS schedulers can co-operate in a large-scale GCE. This is because the ability of resource provision is computed at the sites itself. Therefore, each site provides its willingness to accept a job. As multiple jobs arrive in a site, the Euclidean measure is computed sequentially and resources pre-emptively deducted. These resources will be re-included in the site as a "cache clear" is received for the intended job ID. This ensures that resources are correctly reported at every ACK to the job submission locations. This also provides a starting point for implementing a scalable distributed scheduling mechanism to support a large-scale GCE.
- Independent resource policies can be implemented at every site as the Euclidean measure is calculated within the site. This allows the site administrators to be able to easily define the amount of shared resources available to the GCE without consulting a GCE administrator. Essentially, this reduces the involvement of the administrator in defining "rules" in resource allocation. Again, this reduces the implementation complexity of the MRS system.
- The broadcast and ACK mechanism used in MRS provides a way to identify sites that are disconnected from the site. The timeout function also allows the strategy to discard sites whose resources are possibly more scarce. This helps MRS in identifying sites that it can continue to schedule to even as sites leave and join the Grid environment.

The system and strategy for MRS can be described as a class of Job Sharing strategies operating within a Multi-Site Computing Model [4]. However, when MRS is compared to the models described in [4], clear stages in resource selection and scheduling algorithm does not exist. The computation of the indices combines the selection and scheduling stages and thus reduces the fragmentation of resource considerations during resource allocation and scheduling.

In a strategy wherein resource matching is followed by allocation through a scheduling algorithm, where there are n computing sites in the GCE and m resources to consider, the time complexity of the resource selection stage would be $O(n^m)$. This results in undesirable slow-downs when there are either a huge number of sites, or when there is a large number of resources to consider. The total time is therefore the sum of time-to-allocate and the time-to-schedule.

In MRS, the broadcast of requirements is of time complexity $O(n)$ as each site will only need to receive the resource requirements of a job once. However, due to broadcast, network latencies will be involved, which can possibly lead to slow-downs in MRS. This can be easily prevented by "dropping" sites that do not acknowledge the broadcast in a fixed amount of time. We, thus set an upper limit of the time-to-live (TTL) for each broadcast depending on the network environment MRS is operating in. The worst-case overall time taken for MRS to schedule can thus be written as $2n.TTL+\max(CT_n)$, where $\max(CT_n)$ is the maximum time taken for index computation for a single site. The time complexity therefore remains linear with increase in sites as well as resources when using MRS.

We also investigated the computational complexity of MRS compared to other Job Sharing strategies in a Multi-Site Computing Model. When a strategy separates the resource selection and the scheduling phases, two main components contribute to the computational complexity of the strategy for each job. First, the sorting and filtering methodology used in the resource selection phase, and secondly, the scheduling complexity incurred in the algorithm used. In MRS, the creation of the Virtual Map for each job is essentially a sort of the (x, y) indexes provided by the sites participating in the MRS. This is simplified further when we use the Euclidean distance as a measure of match. The computational complexity is therefore only dependent on the sorting algorithm. This is because scheduling in MRS is a one step process. It is also noted that the computation complexity of the indices provided by the participating sites is linear to the number of resources and the number of dimensions we wish to consider in the Virtual Map. The increase in the number of sites or resources therefore has no effect on the overall allocation strategy provided in MRS, and thus limits the computation complexity to that of the sorting algorithm used in the system. This is unlike other strategies which can still incur computation complexities in the other stages of allocation. In our implementation, the sorting strategy used is a stable merge–sort where the complexity is $O(n \log n)$.

It should be noted that in MRS, resource considerations are not limited to dependencies. Additional requirements or dependencies can be easily added by extending the number of dimensions to be considered within MRS. This does not severely impact the complexity of MRS in both time and computation complexity when compared to other methods.

The broadcast of resource requirements in the GCE is done "all to all" due to the nature of Job Sharing. This is potentially wasteful when jobs have to be rescheduled due to the lack of resources or are delayed for some reason. We reduce the impact of broadcasting by allowing sites in the GCE to cache all requirements of unscheduled jobs upon reception of a broadcast. Subsequent notifications to try to schedule the same jobs will therefore incur much less overheads in communication. A cancellation broadcast was also introduced to notify all participating sites in the GCE to remove an unscheduled job from its cache. This keeps the entire GCE in sync of the jobs that are remaining to be scheduled.

4.1. Simulations, results and discussions

Based on the system design in Section 4, the GCE is simulated in order to ascertain the performance of MRS.

We compare our MRS with the Backfilling strategy (BACKFILL) [6,16] and a job Replication (REP) strategy [14], which is similar to that used in SETI@Home [12]. We make use of similar Job Sharing and Multi-site environment as described earlier such that the intrinsic advantages of the algorithms can be elicited and quantified.

The workload model provided by Song et al. [25] was used as the workload input. The workload profile is shown in Fig. 6. The metrics described in Section 3 were used to quantify the performance and comparison. The results of our experiments are summarized in Table 1 and Fig. 7. The significance of these results are discussed below.

In the simulation model, jobs are allowed to arrive in a stream over a span of three days. The various job requirements are modeled by the information provided in [15,25] and are injected into the simulation model. Data requirements are also additionally generated in order to simulate the need for data to be transported from one location to another in order for a successful computation to take place.

AWT is the average amount of time a job waits in the queue before being executed. This starts from the point of submission to the point when the job begins its transmission to the execution node. In Fig. 7, we have normalized all the performance indicators to the BACKFILL algorithm in order to look at the performance differences of the experiments.

It was noted that in terms of AWT, both REP and MRS significantly outperforms BACKFILL by 40% and 50%, respectively, when run in a distributed environment. This is due to the fact that the backfill algorithm does not allocate jobs in consideration of the data distribution time. The fact that jobs are streaming into the system also accounts for the inability for the algorithm to be able to obtain a good "packing" schedule where resources will be optimized. We observe that the AWT of REP is far better than BACKFILL. This is attributed to the fact that as a job gets replicated, the likelihood of being allocated to a faster resource or bandwidth increases. This is, however, non-optimal as it was achieved without making full use of the information available in the execution environment. This non-optimality is verified by the fact that MRS is able to achieve an even better AWT by making use of inter-resource relationships defined within its indices.

From Table 1, we can also clearly see that the utilization for BACKFILL is the lowest in all the experiments. REP and MRS exhibits increasing levels of utilization which accounts for a shorter AWT. However, it may be noted that in the replication algorithm, every job is essentially submitted twice in order to achieve better performance. This replication potentially hinders the execution of other jobs that might require more CPUs in the GCE. This can also artificially inflate the utilization of the GCE. This is clear from the fact that an increase in utilization using the REP strategy does not lead to any improvement in the QCT. It has, instead, induced a detriment to the GCE by almost 70% when compared to BACKFILL. In contrast, we can see an improvement of 18% when comparing the utilization between MRS and REP. This is also directly reflected in the overall QCT which has improved by 52%.

From our experiments, we observe that replication can lead to a degradation of performance when the entire queue is considered. This is clearly reflected in Fig. 7, where REP is performing 71% slower in QCT when compared to BACKFILL. It is to be noted that the time taken for a job to complete its execution is inclusive of the execution overheads and latencies that is associated with data and computation communications.

In contrast to BACKFILL and REP, our simulations have shown that MRS has been able to achieve a 50% improvement AWT, an 18% improvement over QCT and a 29% improvement in AGU. This is due to the fact that MRS makes use of comparative measures on the benefits of allocation to each node. This is inherent to the algorithm during the process of Virtual Map creation. A lower AWT is very much due to a good allocation decision of the resources when MRS is presented with a queue of jobs. This allows for more jobs to be allocated per unit time, which is reflected clearly in the 18% improvement in QCT over BACKFILL. This is achieved without the over allocation of resources as in REP, giving MRS a 52.3% improvement in QCT when compared to a REP. The matching of resources using the computation and data indexes, also results in a much higher utilization, dispatching jobs to nodes that are able to satisfy the jobs while intelligently deciding which jobs to keep local and which jobs to dispatch.

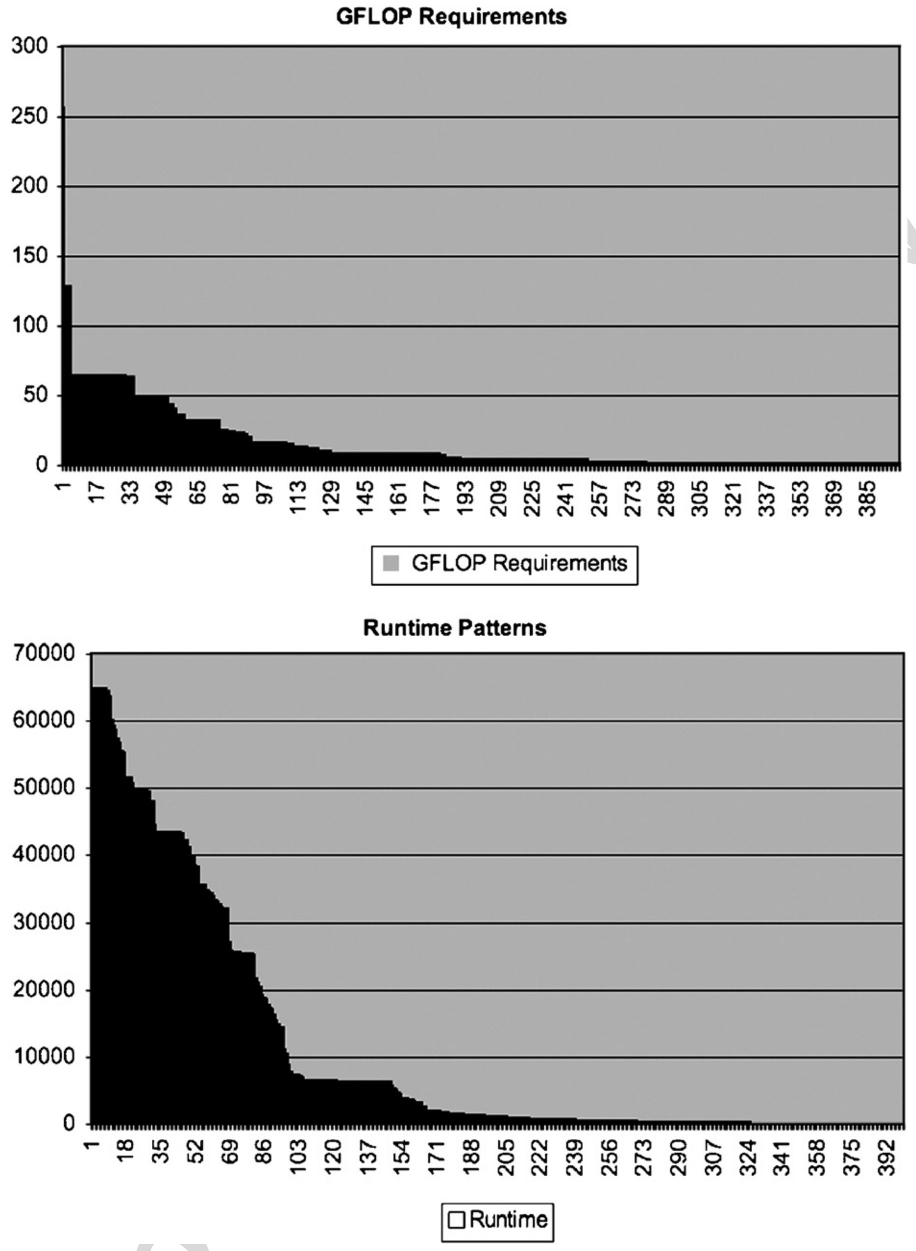

Fig. 6. Workload model profile provided by Song et al. [25].

Table 1
Experimental results comparing BACKFILL, REP and MRS

	AWT (time units)	QCT (time units)	AGU (%)
Tabulated experimental result			
BACKFILL	2579.43	187302.22	57.78
REP	1500.40	320362.16	63.08
MRS	1266.71	152810.98	74.46
	AWT (%)	QCT (%)	AGU (%)
Percentage improvement over BACKFILL			
REP	41.83	(71.04)	9.16
MRS	50.89	18.41	28.86
Percentage improvement over REP			
BACKFILL	(71.91)	41.53	(8.40)
MRS	15.58	52.30	18.04

In view of the workload model used, we observe that many of the jobs in the simulation model requires between 1 and 64 GFLOPs. A majority of the jobs also require run times less than half the longest running job. On comparing this workload model that we are using with those from San Diego Super-Computing Center (SDSC), Lawrence Livermore National Laboratory (LLNL) and Kungliga Tekniska hogkölan - Royal Institute of Technology (KTH), we find that our workload profile exhibited close similarities when compared to [15,10]. This provides further assurance that MRS is able to provide advantages in scheduling when applied to other common workload.

In general, it is observed that MRS is able to render a performance that is much suited for scheduling resources over a GCE.

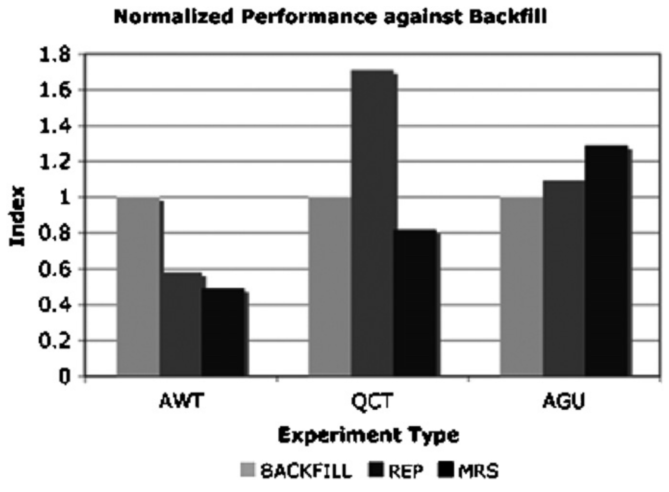

Fig. 7. Normalized comparison of simulation to backfill algorithm.

5. Related work

There have been other strategies introduced to handle resource optimization for jobs submitted over Grids. However, while some investigated strategies to obtain optimizations in the computational time domain, others looked at optimizations in data or I/O domain. Recently, more creative methods to achieve optimal scheduling have included the concept of the costs of resources in financial terms. Some of these techniques, which are relevant to the context of this paper, will be introduced below.

In [26], job optimization is handled by redundantly allocating jobs to multiple sites instead of sending it only to the least loaded site. The rationale in this scheme was that the fastest queue will allow a job to execute before its replicas and this provides low wait times and improves turn-around time. Job allocation failures due to site availabilities would also be better handled due to this redundancy. However, this strategy leads to problems where queue lengths of different sites are unnecessarily loaded handle the same job. The frequent changes in queue length can also potentially hamper on-site scheduling algorithms to work effectively as schedules are typically built by looking ahead in the queue. In addition, the method proposed does not investigate the problems that can arise when the data required for the job is not available at the execution site and needs to be transported for a successful execution. MRS works to eliminate these issues by allocating only the right amount of resources to jobs that requires it, thus freeing up queues from potentially non-executing jobs.

In [33], Zhang has highlighted that the execution profiles of many applications are only known in real-time, which makes it difficult for an "acceptance test" to be carried out. The study also broke down the various scheduling models into (1) centralized, wherein all jobs are submitted at a central location for scheduling and dispatching, (2) decentralized, wherein jobs are submitted at their local locations for dispatching, and (3) hierarchical model, wherein jobs are submitted to a meta-scheduler but are dispatched to low-level schedulers for dispatching and execution. Effective virtualization of resources was also proposed in order to abstract the resource environment and hide the physical boundaries defined. A buddy set as in [23] was also proposed, and its effectiveness also highlighted in [1], where it was shown that when groups of trusted nodes co-operate, the resulting performance is superior compared to situations where there is no relationship establishment between nodes. However, in both cases, the strategies proposed looks plainly at the computational requirements of a job and does not consider the data resource required. It also does not address resource allocation pertaining to both serial and parallel job requirements. MRS effectively applies the concept of co-operation and virtualization to exploit the advantages presented in [1,23], but includes knowledge of bandwidth to account for I/O and communication overheads. While this allows us to apply MRS to both serial and parallel jobs, it also allows us to efficiently schedule in a Grid environment where the data resources are distributed.

In the work presented in [13], the ability to schedule a job in accordance to multiple (K) resources is explored. Although the approach was not designed with the Grid environment in mind, the simulation work presented in [13] shows clearly the potential benefits where scheduling with multiple resources is concerned. Performance gains of up to 50% were achieved when including effective resources awareness in the scheduling algorithm. Similar resource awareness and multi-objective based optimizations were studied in [31]. In both cases, the limitations of inconventional methods was also identified as there was have no mechanism for utilizing additional information known about the system and its environment. However, in [13], there was no data resources identified, while in [31], we believe that the over simplicity of resource aggregation was inadequate in capturing resource relationships. MRS proposes a more complex form of resource aggregation that allows for better expression of resource relationships, while maintaining simplicity in the algorithm construction. At the same time we continue to consider multiple resources which includes both computational and data requirements.

In [22] data replication and reuse of resources was looked into as a means of establishing a Grid being able to handle large data (i.e., Data Grid). Elizeu et al. has looked into the classification of tasks that are processors of huge data (pHD), where by processes require large data sets and data reuse is possible. They introduced a term referred to as Storage Affinity, which takes into account on how reusable is a set of data by pHDs or a bag of tasks. This also determines if a task should be sent to a location where the required data resides or vice versa. Following this, task replication [14] is used to reduce the wait time of the job. This method is useful to handle pre-replicated or re-usable data but does not address how the data would be best scheduled for applications with no reusable data. However, [22] has demonstrated that it is possible to improve response times for jobs through smart data management. We build on this concept of affinity in our algorithm, combined with better resource relationship representation, to arrive at a strategy that would allow the overall overheads of data transmission to be minimized. This is done with no detrimental effect on the wait times of a job and the overall queue completion of the Grid environment.

Contributions in [30] considered the idea of replication and further included a data catalog method to discover and the best location to use. Making use of the Network Weather Service [32], it is possible to determine the best node to collect the data from/send a job to. Then, a compute–data pair is assigned with the earliest completion time. This, method has again identified that data optimization is critical to the response time of a job. This, however, does not exploit resource locality w.r.t. the serial or parallel job requirements. This is thus unsuitable for jobs that are highly parallel in nature (i.e., for applications customized for distributed memory systems). We look upon parallel jobs as applications that requirements low latency and high bandwidth, and assign the resource allocation such that both parallel and serial jobs are optimized.

In [21], Ranganathan et al. presented that Computation Scheduling and Data Scheduling can be considered asynchronously in Data-Intensive Applications. The study considered External Schedulers, local Schedulers and Data schedulers. It concludes that data movement and computation need not always be coupled for consideration together. While this might be true, and demonstrated in [30], through High Energy Physics applications, this is not always the case when MPICH-G2 type applications [11,19] are concerned. MRS recognizes parallel job requirements and, by using affinity and combined resource allocation, decides the best sites for the job to be dispatched to such that everything is in the same path.

Other projects such as the Storage Resource Broker [20] and OGSA-DAI [17] mainly concentrates on assisting the access and integration of data in a distributed computing environment such as a Grid. By itself, these middleware does not decide nor allocate the availability of data resources.

While many other works such as [3,29] continues to provide algorithms to effectively allocate resources, much of these works on the premise of [21] where data and computation resource requirements are handled separately. While these mechanisms are shown to be effective in Monte-Carlo or parameter sweep type applications where the tasks or sub tasks are considered to be independent, we hesitate to generalize on its effectiveness when the nature of jobs, such as MPI-G2 parallel class of applications, can lead to inter-resource dependence. Although many of these algorithms work effectively over a known set of resources, the complexity of the strategies makes it difficult to include additional resources to the Grid. MRS seeks to eliminate this limitation to allow additional resource considerations to be easily added for consideration through aggregation and representation of resource dependence. Our simulation demonstrates this aggregation to cater for data and communication overheads while at the same time, taking care of both requirements of serial and MPI parallel application, especially during fragmentation.

6. Conclusions

In this paper, we have proposed a novel distributed resource scheduling algorithm capable of handling several resources to be catered among jobs that arrive at a Grid system. Our proposed algorithm, referred to as MRS algorithm, takes into account the different resource requirements of different tasks and shown to obtain a minimal execution schedule through efficient management of available Grid resources. We have proposed a model in which the job and resource relations are captured and are used to create an aggregated index. This allows us to introduce the concept of Virtual Map that can be used by the scheduler to efficiently determine a best fit of resources for jobs prior to execution. We also introduced the concept of Resource Potential to identify inter-relations between resources such as bandwidth and data. This allows us to identify sites that has least execution overheads with respect to a job.

In order to quantify the performance, we have used performance measures such as average job wait times, queue completion times, and average resource utilization factor, respectively. We considered practical workload models that are used in real-life systems to quantify the performance of MRS. Performance of MRS has been compared with conventional backfill and replication algorithms that are commonly used in a GCE. Workload models based on recent literature [25] was also used. Our experiments have also conclusively elicited several key performance features of MRS with respect to the backfill and replication algorithms, yielding performances improvements up to 50% on some performance measures.

Below we briefly discuss on some possible immediate extensions to the problem we have addressed in this paper. Although the techniques used in MRS appear to be promising, we believe that it can be extended to include more dimensions then just computational and data. For instance, using the Virtual Map technique, it is possible that other parameters such as, Quality-of-Service [7,28], economic considerations [2] can be included into the model by simply extending the number of dimensions of consideration. These new considerations and how it interacts with other parameters have to be studied carefully to quantify the inter- and intra-resource relationship and then represented into an aggregation equation which can be used in MRS. It would be interesting to consider expanding our simulation environment to include latency information and not assume the direct relation between bandwidth and latency. Lastly, it would be more interesting to invent advanced techniques of job arrangement and fragmentation of jobs to thoroughly exploit the idling resources during the execution of jobs, especially when job queues are insufficient to fully utilize a Grid computing environment.

Acknowledgments

Parts of this work addressed certain objectives of a funded project in Grid Computing under the Grant 052 015 0024/R-263-000-350-592 from National Grid Office via A*Star, Singapore. Bharadwaj Veeravalli would like to thank the funding agency in supporting this research.

References

[1] F. Azzedin, M. Mahewaran, Integrating trust into grid resource management systems, in: Proceedings of ICPP 2002, 2002.

[2] R. Buyya, M. Murshed, D. Abramson, S. Venugopal, Scheduling parameter sweep applications on global grids: a deadline and budget constrained cost–time optimisation algorithm, Int. J. Software: Pract. Exper., This document can also be found at: ⟨http://www.gridbus.org/~raj/cv.html#papersjl⟩.

[3] H. Casanova, A. Legrand, D. Zagorodnov, Heuristics for scheduling parameter sweep applications in grid environments, in: Ninth Heterogeneous Computing Workshop 2000, 2000.

[4] C. Ernemann, V. Hamscher, U. Schwiegelshohn, R. Yahyapour, On advantages of grid computing for parallel job scheduling, in: Proceedings of the Second IEEE/ACM International Symposium on Cluster Computing and the Grid, 2002.

[5] I. Foster, C. Kesselman, The Grid: Blueprint for a New Computing Infrastructure, second ed., Morgan-Kaufman, Los Altos, CA, 2004.

[6] V. Hamscher, U. Schwiegelshohn, A. Streit, Evaluation of job-scheduling strategies for grid computing, in: Proceedings of the First IEEE/ACM International Workshop on Grid Computing, Brisbane, Australia, 2000.

[7] X.S. He, X.H. Sun, G. von Laszewski, QoS guided min–min heuristic for grid task scheduling, J. Comput. Sci. Technol. 18 (4) (2003) 442–451.

[8] Hewlett Packard, High Performance Technical Computing, ⟨http://www.hp.com/techservers⟩, 2004.

[9] IBM, Cluster Servers, ⟨http://www-1.ibm.com/servers/eserver/clusters/⟩, 2004.

[10] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, J. Riodan, Modeling of workload in MPPs, in: D.G. Feitelson, L. Rudolph (Eds.), Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science, vol. 1291, Springer, Berlin, 1997, pp. 95–116.

[11] N. Karonis, B. Toonen, I. Foster, MPICH-G2: a grid-enabled implementation of the message passing interface, J. Parallel Distrib. Comput. (JPDC) 63 (5) (2003) 551–563.

[12] E. Korpela, D. Werthimer, D. Anderson, J. Cobb, M. Lebofsky, SETI@home—massively distributed computing for SETI, Comput. Sci. Eng. 3 (1) (2001) 81.

[13] W. Leinberger, G. Karypis, V. Kumar, Job scheduling in the presence of multiple resource requirements, in: Proceedings of the IEEE/ACM SC99 Conference, Portland, Oregon, USA, November 13–18, 1999, pp. 47–48.

[14] Y. Li, M. Mascagni, Improving performance via computational replication on a large-scale computational grid, in: IEEE/ACM CCGRID2003, Tokyo, 2003.

[15] U. Lublin, D.G. Feitelson, The workload on parallel supercomputers: modeling the characteristics of rigid jobs, Technical Report 2001-12, School of Computer Science and Engineering, The Hebrew University of Jerusalem, October 2001.

[16] A.W. Mu'alem, D.G. Feitelson, Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling, IEEE Trans. Parallel Distrib. Syst. 12 (6) (2001) 529–543.

[17] Open Grid Service Architecture Data Access and Integration, ⟨http://www.ogsadai.org.uk/⟩.

[18] Parallel Workload Archive: Models, ⟨http://www.cs.huji.ac.il/labs/parallel/workload/models.html⟩.

[19] K.-L. Park, H.-J. Lee, O.-Y. Kwon, S.-Y. Park, H.-W. Park, S.-D. Kim, Design and implementation of a dynamic communication MPI library for the grid, Int. J. Comput. Appl. 26 (3) (2004) 165–171.

[20] A. Rajasekar, M. Wan, R. Moore, W. Schroeder, G. Kremenek, A. Jagatheesan, C. Cowart, B. Zhu, S.-Y. Chen, R. Olschanowsky, Storage resource broker—managing distributed data in a grid, Comput. Soc. India J. 33 (4) (2003) 42–54 (special issue on SAN).

[21] K. Ranganathan, I. Foster, Decoupling computation and data scheduling in distributed data-intensive applications, in: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing HPDC-11 (HPDC'02), Edinburgh, Scotland, July 24–26, 2002, pp. 352–358.

[22] E. Santos-Neto, W. Cirne, F. Brasileiro, A. Lima, Exploiting replication and data reuse to efficiently schedule data-intensive applications on grids, in: Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing, June 2004.

[23] K.G. Shin, Y. Chang, Load sharing in distributed real-time systems with state change broadcasts, IEEE Trans. Comput. 38 (8) (1989) 1124–1142.

[24] N. Snyder, IBM Linux Clusters, ⟨http://linux.ittoolbox.com/documents/document.asp?i = 2042⟩, 2002.

[25] B. Song, C. Ernemann, R. Yahyapour, User group-based workload analysis and modelling, Cluster and Computing Grid Workshop 2005, Cardiff, UK, 2005.

[26] V. Subramani, R. Kettimuthu, S. Srinivasan, P. Sadayappan, Distributed job scheduling on computational grids using multiple simultaneous requests, in: Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing HPDC-11 20002 (HPDC?02), Edinburgh, Scotland, July 24–26, 2002, pp. 359–368.

[27] Sun Microsystems, High Performance Technical Computing, ⟨http://www.sun.com/solutions/hpc⟩, 2004.

[28] A. Takefusa, H. Casanova, S. Matsuoka, F. Berman, A study of deadline scheduling for client-server systems on the computational grid, in: Proceedings of the 10th IEEE International Symposium on High Performance Distributed Computing (HPDC-10), 2001, pp. 406–415.

[29] K. Taura, A. Chien, A heuristic algorithm for mapping communicating tasks on heterogeneous resources, in: Ninth Heterogeneous Computing Workshop 2000, 2000.

[30] S. Venugopal, R. Buyya, L. Winton, A grid service broker for scheduling distributed data-oriented applications on global grids, Technical Report, CoRR cs.DC/0405023, 2004, This can be located at: ⟨http://www.gridbus.com⟩.

[31] K.N. Vijay, L. Chuang, L. Yang, J. Wagner, On-line resource matching for heterogeneous grid environments, in: Cluster and Computing Grid Workshop, Cardiff, UK, 2005.

[32] R. Wolski, G. Obertelli, Network Weather Service, ⟨http://nws.cs.ucsb.edu⟩, 2003.

[33] L. Zhang, Scheduling algorithm for real-time applications in grid environment, in: Proceedings on IEEE International Conference on Systems, Man and Cybernetics, vol. 5, USA, 2002.

Khoo Boon Tat, Benjamin, has been involved with High Performance Computing since 2000, and received his B.Eng. from the National University of Singapore in 2001. Prior to this, he has worked in the area of High Performance Computing with IBM and has implemented multiple research centric as well as commercial Grid infrastructures in the Asia Pacific region. He was also part of the Institute of High Performance Computing in Singapore (2003), as part of a research team involved in Grid architectures and middleware. Since then, he has implemented many more clusters and Grids including visualization clusters and distributed computing infrastructures on many platforms. He is now in Apple Inc., as a senior consultant in enterprise solutions and scientific markets for South Asia. He is also currently pursuing his M.Eng. from the National University of Singapore. His research interest includes distributed scheduling for cluster/Grids, distributed data management and large-scale visualization systems.

Bharadwaj Veeravalli (http://cnds.ece.nus.edu.sg/elebv), received his Ph.D. from the Department of Aerospace Engineering, Indian Institute of Science (IISc), Bangalore, India, in 1994, Masters in Electrical Communication Engineering from IISc, Bangalore, India in 1991 and B.Sc. in Physics, from the Madurai-Kamaraj University, India, in 1987. He did his post-doctoral research in the Department of Computer Science, Concordia University, Montreal, Canada, in 1996. He is currently with the Department of Electrical and Computer Engineering, at The National University of Singapore, as an Associate Professor. His research interests include, Cluster/Grid computing, Scheduling in Parallel and Distributed systems, Bioinformatics, and Multimedia computing. He has published extensively in international journals and conferences, and has co-authored three research monographs in the areas of Parallel and Distributed Systems, Distributed Databases, and Networked Multimedia Systems. He is serving the editorial board of IEEE TC, IEEE TSMC-A, IJCA, USA, as an Associate Editor.

Dr. Terence Hung is a Programme Manager at the Institute of High Performance Computing (Singapore) with an adjunct associate professorship at the Nanyang Technological University. He leads research in grid computing, HPC, data mining and visualization. Terence sits on various committees to drive national level grid activities. He has served as external proposal reviewer for NSERC (Canada) and consultant to Gerson Lehrman Group on HPC. Terence's research interests include efficient multi-core algorithms, adaptive parallelism and software as a service. He is PI/co-PI in various local and international grant projects. Terence has a Ph.D. in Electrical Engineering from the University of Illinois at Urbana-Champaign.

Dr. Simon See is currently the High Performance and Grid Computing Technology Director for Sun Microsystems Inc. and also the Director/Chief Technologist for Sun Asia Pacific Science and Technology Center. He is also an Adjunct Associate Professor in both the Nanyang Technological University and the National University of Singapore. His research interest is in the area of High Performance Computing, Computational Science, Applied Mathematics and Simulation Methodology. He has published over 50 papers in these areas and has won various awards. Dr. See graduated from the University of Salford (UK) with a Ph.D. in Electrical Engineering and Numerical Analysis in 1993. Prior to joining Sun, Dr. See worked for SGI, DSO National Lab. of Singapore, IBM and International Simulation Ltd (UK). He is also providing consultancy to a number of national research and supercomputing centers.