An Evaluation of Job Scheduling Strategies for Divisible Loads on Grid Platforms

Yudith Cardinale
Dpto. de Computación y Tecnología de la Información
Universidad Simón Bolívar
Apartado 89000, Caracas 1080-A, Venezuela
Email: yudith@ldc.usb.ve

Henri Casanova
Dept. of Information and Computer Sciences
University of Hawai'i at Manoa
1680 East-West Road, Honolulu, HI 96822, U.S.A.
Email: henric@hawaii.edu

ABSTRACT

In this paper we study *distributed job scheduling* in grid environments when each job is a *DL application*. The scheduling goal is to minimize the average steady-state job turnaround time. In this context, we identify in which regimes classes of scheduling strategies are efficient, namely for which platforms and which communication to computation ratios. We also quantify what level of global information about the platform is required for efficient scheduling. All our findings are obtained via simulation of wide ranges of application and platform scenarios. Our most significant findings are that the use of grid information is only necessary at high workload, and that at high workload using dynamic information improves performance by around 10% when compared to using static information.

I. INTRODUCTION

Improvements in wide-area networking infrastructure and middleware technology have enabled the aggregation of resources from multiple administrative domains into grid platforms. These platforms provide unprecedented capabilities to applications, but only if resources can be used effectively. As a result, a popular research question is: How to assign components of a *single* distributed application to grid resources in order to optimize some metric (e.g., turnaround time)? Another important question is that of scheduling multiple independent applications, or *jobs*, so that some aggregate metric is optimized (e.g., average turnaround time). In this paper we focus on the latter question. Job scheduling has been addressed from a practical standpoint in the context of single parallel resources with *batch schedulers* [12]. While batch schedulers have been used for several decades, they typically take a resource-oriented view of scheduling and do not attempt to optimize user-centric metrics, such as average turnaround time. From the theoretical standpoint, several researchers have studied the question of job scheduling with a user-centric metric in mind [3]. All these works focus on homogeneous systems and on centralized scheduling.

By contrast with the aforementioned works, we focus on the problem of distributed job scheduling in heterogeneous grid environments consisting on multiple sites where each site has a homogeneous parallel compute platform managed by its own job scheduler, and with a focus on an aggregate user-centric metric (in our case the average turnaround time across all jobs). Job scheduling over a multi-site grid has often been refereed to as *meta-scheduling*, and has been studied by many researchers [15], [27], [31], [14], [17], [26], [29], [7], [8]. The works in [15] and [27] distinguish between centralized and decentralized meta-scheduling schemes. In the centralized scheme [31], [14], [17] a single scheduler is used to make all scheduling decisions, which limits scalability and is a single point of failure. Our work focuses on decentralized meta-scheduling and is thus most related to [26], [29], [7], [8]. Our system model is inspired by the one in [26]. Subramani et al. [29] describe distributed scheduling algorithms that use multiple simultaneous requests. Our approach does not use multiple simultaneous requests and differs from both [26] and [29] in that we allow for jobs to be split among multiple sites, or "co-scheduled". This co-scheduling approach is also taken in [7], [8]. However, our work employs a different application model, which has major implication on meta-scheduling and which we introduce below.

The work in [7], [8] and other works that consider co-scheduling target data-parallel applications, generally requiring a resource reservation infrastructure as application processes must run concurrently, typically leading to frequent communications over the wide-area. This is certainly feasible in grid platforms that use proprietary networks, but more difficult to justify on grids that use the Internet for communications and thus can be largely impacted by (variations in) network delays. However, there is an entire class of relevant scientific applications that are amenable to wide deployments over such networks: *divisible load (DL) applications* [5]. These applications consist of an amount of work that can be arbitrarily divided into *chunks*. These chunks can then be sent to remote resources and can be computed independently with no necessary synchronization. In spite of its simplicity, the DL model arises in many fields of science and engineering [10],

[24], [16], [13], [1], and the issue of DL scheduling has been addressed by many authors (see [2] for a summary of current results). These applications are ideal candidates for co-scheduling on grids that consist of multiple sites over the wide-area [6].

We study the problem of *job scheduling* in grid environments when each job is a *DL application*. The scheduling goal is to minimize the average steady-state job turnaround time. In this context, we identify in which regimes classes of scheduling strategies are efficient, namely for which platforms and which communication to computation ratios. We also quantify what level of global information about the entire platform is required for efficient scheduling. This is a critical point for determining what type of information and monitoring infrastructure is needed for effective meta-scheduling in grid environments. All our findings are obtained via simulation.

II. MODELS

A. Platform Model

We consider a platform that consists of N *sites*, indexed by $i = 1, \ldots, N$, connected over the wide-area. We model the wide-area network as a fully connected network. Each site reaches the WAN through a LAN link (we assume that all such LAN links have the same bandwidth). The bandwidth on each LAN link is shared among each flow going through the link. The bandwidth on WAN link is not shared and each flow going through the link gets the full bandwidth. In other terms, we do not model any contention between the flows of *our applications* on the wide-area. This model is justified by the bandwidth-sharing properties observed on wide-area links: when such a link is a bottleneck for an end-to-end TCP flow, several extra flows can generally be opened on the same path and they each receive the same amount of bandwidth as the original flow. This behavior can be due to TCP itself (e.g., congestion windows), or to the fact that the number of flows belonging to a single application is typically insignificant when compared to the total number of flows going through these links. This property is often exploited by explicitly opening parallel TCP connections and we have observed it in our own measurements [9]. Some recent work [18], [9] provides the foundation for refining our network model, both based on empirical measurements and on theoretical modeling of network traffic, and we leave such developments for a future paper.

Figure 1 depicts our platform model. Each site comprises a number of *compute resources*, np_i, as well as storage, which we assume to be unlimited. As in [26] compute resources are managed by a *Local Scheduler* (LS) and requests for computations are managed by an *External Scheduler* (ES). Local users submit requests to the ES directly. Then, for each request, the ES decides whether to send the request to a remote LS or to send it to the local LS. The LS assigns applications to local compute resources in a FIFO fashion. Note that real-world systems use batch schedulers that typically employ some backfilling scheduling algorithm [12], which is quite different from FIFO. However assuming FIFO local scheduling is

done in many previous meta-scheduling work [15], [27], [14], [26], [29], [7], [8] because it makes it easier to understand and analyze the behavior of the meta-scheduling algorithms. Furthermore, we will see below that we simulation DL job that run on all resources at a site, making backfilling unnecessary and FIFO scheduling a reasonable approach.
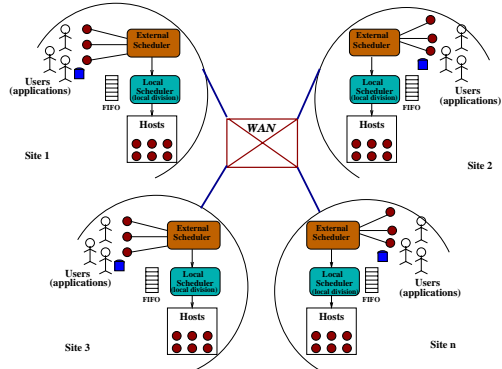

Fig. 1. Grid Model

By contrast with [26], our model allows the ES to *divide* each application into *chunks* and assign these chunks to the LS at the local site as well as to the LS at one or more remote sites. This is because we consider DL applications, which allow asynchronous parallel executions. Another consequence of considering DL applications is that we assume that each chunk can utilize all resources at the site to which it is assigned: each chunk is itself a DL application and if a site consists of R resources, a chunk is divided into R *sub-chunks*. This is a strong assumption as there is always overhead involved for using increasing numbers of processors, even for DL applications. Therefore, it may be more efficient to use fewer sub-chunks. However, our goal in this paper is to compare and evaluate the effectiveness of meta-scheduling policies, which are orthogonal to the application scheduling policies implemented at the LS level. Consequently, we leave this issue aside for now but discuss possible evolutions of our model in Section III. Also, we ignore the cost of local I/O for application data. This cost could be easily integrated into our model (i.e., as a proportional overhead), but once again this issue is orthogonal to that of meta-scheduling. Finally, we assume that the LSs can accommodate for possible heterogeneity among the compute resources at a site for each chunk execution via load balancing. Such load balancing is straightforward because each chunk can be divided into sub-chunks of arbitrary size to account for different processor speeds (since the load is divisible) and because we ignore the cost of local I/O. In effect, load balancing can negate the effect of heterogeneity. A consequence of the above consideration is that we can restrict our experiments to homogeneous compute resources, but with different numbers of resources at each site.

B. Application and Workload Model

We consider applications that consist of a continuous load that can be arbitrarily divided among compute resources. Each

application is described by the size of the input application data in bytes, W_{total}. The amount of processing power needed per byte of load is denoted by $comp\_factor$. The computation time required to process the entire load, $comp\_cost$, is then:

$$comp\_cost = \frac{W_{total} * comp\_factor}{processor\_speed},$$

where $processor\_speed$ is the speed of the processor processing the load.

The time to transfer the load from one site to another is defined as:

$$transfer\_cost = \frac{W_{total}}{bandwidth},$$

where $bandwidth$ is the data transfer rate on the network. Note that, like many other authors [26], [22], [29], we ignore output data transfers: since we are interested in the *steady-state behavior* of the system, transfers of output data can easily be modeled by increasing the size of the input data.

The communication to computation ratio of an application is defined as:

$$comm\_comp\_ratio = \frac{transfer\_cost}{comp\_cost}$$

Note that this ratio is for a given application running on a given platform: the application execution on this platform spends $comm\_comp\_ratio$ times more seconds communicating than computing. Typically, this ratio is much lower than 1 so as to justify the use of distributed resources in the first place. For high such ratios it is typically more advantageous to have the application execute solely on its local site. Our experiments explore ranges of $comm\_comp\_ratio$ values.

Since the load, W_{total}, is divisible, the scheduler can decide how big a chunk of the load to give out to each site. For a platform with N sites, a portion of the load $\alpha_i \times W_{total}$ is assigned to each site with the obvious constraints:

$$\forall i \quad 0 \leq \alpha_i \leq 1 \quad \text{and} \quad \sum_{i=1}^{N} \alpha_i = 1.$$

We model the workload of the entire system as streams of jobs arriving at each site according to a Poisson process with rate λ. While this model does not correspond to a particular workload in the real-world, it is convenient to obtain a first order evaluation of our meta-scheduling heuristics and is easily parameterized by λ. The alternative would have been to use logs from HPC systems such as the ones available at [20], but these logs are for specific systems and user communities. Workload characterization is notably arduous [11], but recently the work in [21] has proposed a new model for job arrival times. We plan to use this model in future work to see if there is any impact on our simulations. However, these logs and models are for rigid parallel jobs and it is not clear whether their arrival times would be representative of a workload consisting of DL jobs.

III. Scheduling Divisible Load Jobs

A. Scheduling Scenario

The problem we address is that of minimizing the average turnaround time over all jobs in the system, when the system is in steady-state. Our scheduling scenario is as follows. At each site users submit jobs (i.e., requests to compute a DL) to the ES throughout time. For each job, the ES determines the α_i values, that is which sites will participate in the application execution and how the load will be divided into chunks among these sites. The chunks are then sent to the LSs at the chosen sites over the network. The LSs divide each chunk by the number of resources, into *sub-chunks*. The turnaround time is defined as the time between the instant at which the job submission arrives to the ES and the instant at which all chunks have been processed by all selected sites.

B. Scheduling Techniques

In order to make decisions, the ESs can make use of information about the grid platform. We consider two types of resource information: the (nearly) static characteristics of the sites (number of processors and their speed); and dynamic characteristics (the length of the FIFO queues).

The allocation process of the ESs consists of two parts, the *selection* of the sites and the *division* of the load among those sites. The selection decides which α_i values will be non-zero, whereas the division assigns actual values to the non-zero α_i's. Based on these considerations we instantiate three classes of meta-scheduling strategies.

1) **Basic Strategies (B):** These strategies make no use of grid information and operate with three flavors of the selection strategy.

Local site (B-LS): The ES does not divide the load and only uses the local site.

All sites (B-AS): The ES divides the load equally among all sites.

Local site and k random sites (B-LkR): Additional to the local site, the ES selects k random and divides the load randomly among the k + 1 sites.

2) **Strategies using static information (Load Balancing 1 - LB1):** Load is divided based on static grid information, proportionally to the number of processors at each selected site.

All sites (LB1-AS): All sites participate in each execution.
Local site and k random remote sites (LB1-LkR): The ES selects k + 1 sites, the local one and k random remote sites.

3) **Strategies using dynamic information (Load Balancing 2 - LB2):** Each ES keeps track of the length of FIFO queues at all sites, and load is divided based in queue length at each selected site.

All sites (LB2-AS): All sites participate in each execution.
Local site and k random remote sites (LB2-LkR): The ES selects k + 1 sites, the local one and k random remote sites.
Local site and k best remote sites (LB2-LkB): The ES uses the local site and k *best* remote sites. The best sites sites are defined as the ones having the highest $power\_ratio$ value defined as:

$$power\_ratio_j = \frac{np_j}{queue\_size_j}, j \neq i,$$

where $queue\_size_j$ is the size of the FIFO queue (i.e., number of pending jobs) in the LS's queue at site j.

A. Bucur et al. [8] propose placement policies based on the knowledge of the number of idle processors in each site and distribute the tasks in such a way as to keep the load balanced. Our LB2 strategies are related to the policies in [8] but the dynamic resource information we consider here is the size of the queues at each LS. Also, the selection strategy used in **LB1-L**k**R**, **LB2-L**k**R**, and **LB2-L**k**B** is loosely related to the **K-Distributed Model** proposed in [29], which chooses a limited number of sites for replicating requests for computations. Our work differs in that we take into account data transfers and we perform co-scheduling. The load distribution strategies proposed in [22] are based on static and dynamic information. Each site receives a *chunk* that is proportional to its computing speed if it has enough free buffer capacity; otherwise that site will be no considered in the current selection.

Other strategies are possible and in total we have evaluated twenty strategies. Since these strategies all rely on heuristics, it is difficult to compare them analytically. However, our simulations showed that several of the twenty strategies exhibit in practice virtually identical performance. In these cases we have included the simpler strategies in this paper.

IV. EXPERIMENTAL METHODOLOGY

Our experiments investigate the following three questions pertaining to the *division* and *selection* concepts that were highlighted in Section III:

(i) *When are the* **basic strategies** *good enough?* Recall that the basic *division* and *selection* policies do not make use of any information about the system state (the ESs operate in isolation).

(ii) *What is the usefulness of global system information for the division policy?* Note that using up-to-date global information may be costly and may lead to prohibitive overhead.

(iii) *How many and which remote sites should be picked by the selection policy?* Site selection has a direct impact on application data transfer costs and sites must be selected judiciously to avoid network bottlenecks.

Experiments on a real-world testbed would be prohibitive both in terms of time, because they would be limited to a specific configuration, and because they would hardly be repeatable. Instead, we have implemented a simulator using the SIMGRID [18] toolkit, which provides the needed abstractions and realistic models for the simulation of processes interacting over a network.

A. Application Scenarios

We chose the following representative applications to instantiate the job mix used in our simulations (see Table I for details):

BLAST® [13] – A set of search programs designed to identify similarities between biological sequences. Typical data sizes are between 400MB and 2GB and the *comm_comp_ratio* is high (1.0 for a 1GHz P4 and bandwidth 100Mbps).
HMMER [16] – A suite of programs that compare biological sequences using Hidden Markov Models. Typical data sizes are between 400MB and 2GB and the *comm_comp_ratio* is low (0.35 for a 1GHz P4 and bandwidth 100Mbps).
Rendering [25] – A generic rendering program that constructs a sequence of frames based on a scene's geometry. The speed of the computation is dependent on the complexity of the scene and on hardware technology. Furthermore, there are many different rendering techniques. We consider here an image-based algorithm (mosaicking) with a high *comm_comp_ratio* (0.5 for a 1GHz P4 and bandwidth 100Mbps) and we consider data sizes between 4MB and 10MB.
Ray Tracing [28] – A popular technique to generate photo-realistic images. Because of recursion in the algorithm and the possibly large number of rays that may be cast, this program has lower *comm_comp_ratio* than Rendering (0.15 for a 1GHz P4 and bandwidth 100Mbps) with data sizes between 4MB and 12MB.
Volume Rendering [19] – A program to visualize large quantities of volume data. It has a very low *comm_comp_ratio* (0.2 for a 1GHz P4 and bandwidth 100Mbps) with data sizes between 400MB and 1GB.
Synthetic Applications – We also simulated several synthetic applications with different W_{total} and *comm_comp_ratio* in order to cover a wide range of applications.

B. Platforms

We simulate the following platforms (see Table II for full details):
GrADS testbed [4] – composed of 3 clusters at the University of Tennessee, Knoxville (UTK), the University of Illinois, Urbana-Champaign (UIUC), and the University of California, San Diego (UCSD); we assume 10Mb/s bandwidth.
TeraGrid [30] – we consider 4 of the TeraGrid clusters located at the San Diego Supercomputing Center (SDSC), the National Center for Supercomputing Applications (NCSA), the California Institute of Technology (Caltech), and the Argonne National Laboratory (ANL). We assume 10Gb/s bandwidth.
Synthetic Grids – with 2, 4, 6, 8, 10, 12, 14, 16, 18, and 20 sites, with clock rates of 250, 500, and 1000, WAN bandwidths of 10, 100, or 500 Mbps and LAN bandwidth of 100, 500, or 1000 Mbps. We simulate platforms with low, medium, and high heterogeneity, in which the number of processors per site is selected randomly between a and b according to uniform distributions, which is denoted as U[a,b] in Table II. We also performed experiments for a fixed total number of processors in the system and just varying the heterogeneity (measured by the standard deviation of the number of processors at each site).

C. Job Arrivals

Jobs arrive in the system according to a Poisson distribution with parameter λ. The same Poisson distribution was assigned

Application	BLAST	HMMER	Rendering	Ray Tracing	Volume Rendering	Synthetic App
W_{total} (MB)	700	700	10	12	800	10,40,50,100,200
$comp\_factor$	5	15	10	35	25	100,50,7.5,17.5,20

TABLE I

APPLICATION CHARACTERISTICS

Platform	# of sites	# of processors per site	Processor speed	Bandwidth
GrADS	3	(UTK) 8 (UCSD) 4 and 2 (UIUC) 6 and 4	550MHz 400MHz and 450MHz 266MHz and 450MHz	100Mbps (LAN) 10Mbps (WAN)
TeraGrid	4	(SDSC) 128, (NSSA) 256 (Caltech) 64 (ANL) 96 and 16	1GHz 1GHz 2.4GHz and 1GHz	10GbE (LAN) 30GbE (WAN)
Synthetic Grids 1	2, 4, 6, 8, 10, 12, 14, 16, 18, 20	U[2,5] (low heterog.) U[2,10](medium heterog.) U[2,24] (high heterog.)	250MHz, 500MHz, or 1GHz	100, 500 or 1000 Mbps (LAN) 10, 100 or 500 Mbps (WAN)
Synthetic Grids 2	4, 8	randomly selected, keeping 64 total proc. in system	250MHz	500 or 1000 Mbps (LAN) 100 or 500 Mbps (WAN)

TABLE II

SUMMARY OF TESTBED CHARACTERISTICS.

to each site in all experiments. Indeed, we choose to keep the workload homogeneous among sites but to make the compute power heterogeneous (in terms of number of processors at each site). The λ values used in our experiments were 0.3,0.6,0.9,1.2,1.8,2.4,3.0,3.6,4.2,4.8.

V. EXPERIMENTAL RESULTS

We used the simulator for 3 sets of experiments: (i) experiments on GrADS and TeraGrid platforms, (ii) experiments on synthetic platforms, using uniform distributions to select the number of processors on each site, and (iii) experiments on synthetic platforms with a fixed number of processors in the system. We evaluate the job scheduling strategies using average job turnaround times. Each simulation experiment was repeated 10 times and we present averages.

A. Real platforms: GrADS and TeraGrid

Figure 2 shows the average turnaround time for the three applications with the lowest $comm\_comp\_ratio$ on the GrADS platform. The **B-LS** strategy yields the best performance in almost all cases with an improvement between 8% and 79% over the nearest strategy. For with the synthetic application with $comm\_comp\_ratio = 0.044$ and $\lambda = 4.8$, **LB2** strategies using 2 sites (**LB2-L1R and LB2-L1B**) are better than **B-LS** by approximately 5%. Another fact to be noted in this last case is that strategies using 2 sites are better than strategies using all sites. This is because when using all sites the network is overloaded. From these observations, we can conclude that in the GrADS platform it is useful to use a (single) remote site only when the total system workload is high. Nevertheless, it is still interesting to look at other strategies in order to compare them, based on the three proposed questions:

- *When are the **basic strategies** good enough?* The basic strategies using more than 1 site (**B-L1R** and **B-L2R**) become better than the **LB1** and **LB2** strategies as $comm\_comp\_ratio$

and λ decrease. This is when the job have high computational demands but the total workload in the system is low.

- *What is the usefulness of global information for the division policy?* The **LB2** strategies become equivalent or slightly better than **LB1** strategies as $comm\_comp\_ratio$ decreases and as λ increases. This result shows that dynamic information could marginally improve the total performance in the GrADS platform only when the system has high total load.

- *How many and which remote sites should be picked by the selection policy?* Strategies using all sites become worse than using 2 sites as $comm\_comp\_ratio$ decreases and as λ, and thus the overall workload, increases. Regarding *which sites*, picking random sites achieves best or close to best performance.
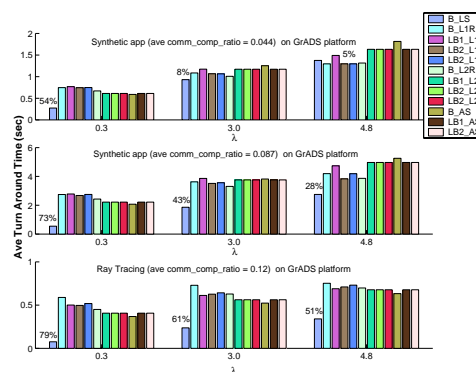

Fig. 2. Average turnaround time on GrADS platform

Figures 3, 4, 5, and 6 show the average turnaround time and the percent of improvement of the best strategy over its nearest, for all applications on the TeraGrid platform with three different λ values. The results show that as the network is faster, using only local resources is rarely judicious. **B-LS**
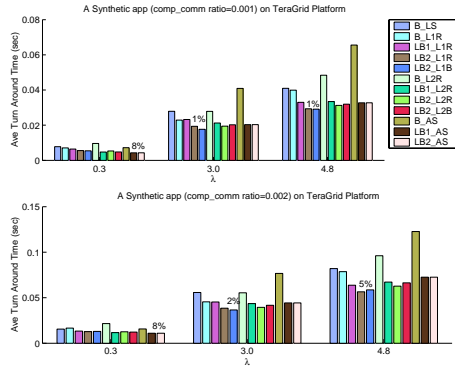
Fig. 3. Average turnaround time for synthetic applications on the TeraGrid platform.
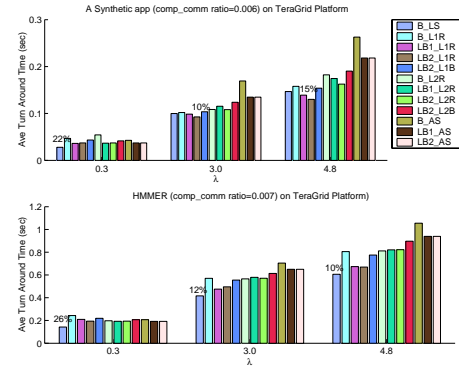

Fig. 5. Average turnaround time for a synthetic application and HMMER on TeraGrid platform

is only the best only when $comm\_comp\_ratio \geq 0.007$ (no matter what λ is), with improvements from 10% to 64% (see Figures 5 and 6). This happen when the job's computational demands decrease. With $comm\_comp\_ratio < 0.007$ it is beneficial to use remote resources. With $\lambda = 0.3$ the strategies **LB1-AS** and **LB2-AS** have the best performance with a low improvement over **LB1-L2R** (see Figures 3 and 4). With $\lambda = 3.0$ and 4.8 the best strategies are that use 2 sites and queue size information (**LB2-L1R** and **LB2-L1B**).

Let us examine the answers to our three main questions:
- *When are the* **basic strategies** *good enough?* For $comm\_comp\_ratio \geq 0.007$, the basic strategy using local and two random sites (**B-L2R**) becomes better than **LB1-L2R** and **LB2-L2R** as the jobs' computational demands decrease. This is the only case in which the basic strategies are better than the other two.
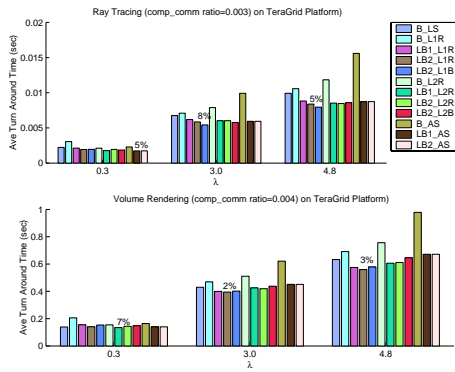

Fig. 4. Average turnaround time for Ray Tracing and Volume Rendering on TeraGrid platform

- *What is the usefulness of global system information for the division policy?* The **LB2** strategies have the same or better performance than **LB1** strategies, except when the $comm\_comp\_ratio \geq 0.01$. Therefore, dynamic information could improve total performance at high total workload.
- *How many and which remote sites should be picked by the selection policy?* Using all sites, with **LB1** and **LB2** strategies, leads to the best performance for $\lambda =$

0.3 and $comm\_comp\_ratio \leq 0.003$. However, when $comm\_comp\_ratio$ and λ increase their performance degrades. In these cases, using the local and a single other site results in better performance than using 3 or all sites. Regarding *which sites* are the best selection, picking random sites results in a performance better, equal or very close to picking the best ones (compare **LB2-L1R** with **LB2-L1B**, and **LB2-L2R** with **LB2-L2B** in figures 3, 4, 5, and 6).

B. Synthetic environments

Now, we turn our attention to synthetic platforms. With these results, we also analyzed the three questions, starting with: *When are the* **basic strategies** *good enough?*.
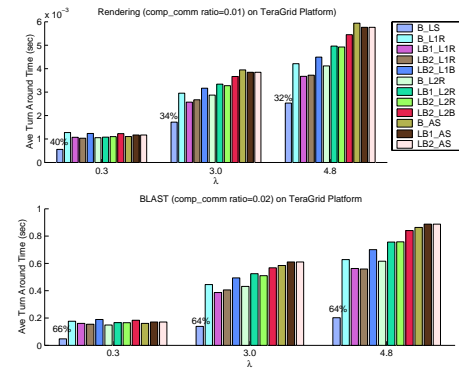

Fig. 6. Average turnaround time for Rendering and BLAST on TeraGrid platform

Figure 7 shows the areas where the **B-LS** strategy is best (shown in black) in terms of $comm\_comp\_ratio$ and of total workload (defined as λ times W_{total}), on average. According to these results using remote resources is beneficial when when $comm\_comp\_ratio < 0.175$ and $\lambda * W_{Total} > 3 * 10^6 bytes$; otherwise the **B-LS** strategy yields the best performance with improvement between 10% and 90%. The **B-AS** strategy yields good performance when the total workload in the system is low, but with high workload it performs the worst.

In most cases (68%), the strategies with the best performance in the white area of Figure 7 were the **LB1-AS** and the

LB2-AS strategies (with almost similar performance). In 27% of cases, best performance was reached when using 2 sites. Using 3 sites led to best performance in 5% of cases.

Figure 7 also shows the areas where the real applications lie. Note that with low $comm\_comp\_ratio$, for all of them, is beneficial to use remote sites.
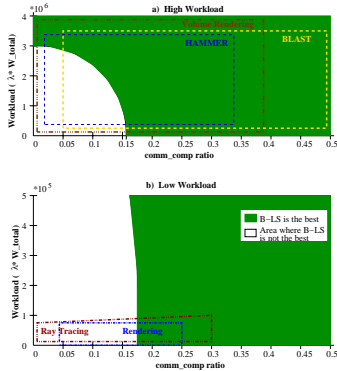

Fig. 7. When is it beneficial (shown in black) to use remote resources given the overall system workload and $comm\_comp\_ratio$ of applications?

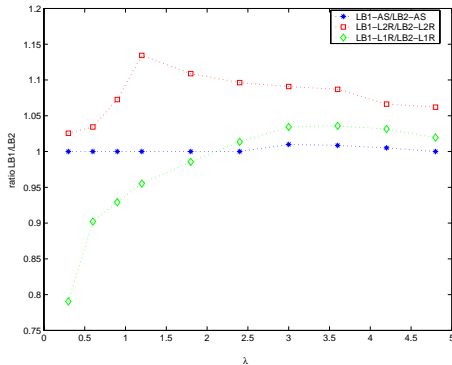*C. Impact of λ on the usefulness of global system information*

Fig. 8. Comparison between **LB1** and **LB2** strategies in a grid platform with 8 sites, 64 total processors, and moderate heterogeneity. $comm\_comp\_ratio = 0.0025$

To answer our second and third questions, we performed a third set of experiments in which we varied λ and the heterogeneity in the system while keeping the number of sites and the number of total processors in the system fixed. When λ increases the overall workload increases as well. We find that the information about FIFO queue sizes becomes beneficial when λ increase, but not when all sites are used. Figure 8 shows an example of this fact on a grid platform with 8 sites, 64 processors in the system, and middle heterogeneity for an application with $comm\_comp\_ratio = 0.0025$. The **LB2-L2R** strategy (using 3 sites) is always better than **LB1-L2R** but the improvement decreases as λ increases. **LB1-L1R** is better than **LB2-L1R** for low λ, but **LB2-L1R** becomes better as λ increases. Indeed, when the arrival rate is high, considering queue size becomes critical as some sites can become more

overloaded than others. Note that when all sites are used, it makes no difference whether queue information is used or not.

We also find that when λ increases, the improvement of using all sites over strategies that use 2 or 3 sites decreases. The reason for this is that the network is overloaded with many transfers of chunks. Figure 9 presents a comparison between **LB2-AS** and strategies using 2 or 3 sites.

D. Impact of heterogeneity in the selection policy

We define heterogeneity as the standard deviation of the number of processors on each site. So a heterogeneity of 0 represents an homogeneous grid platform, in which all sites have the same number of processors. We observe that when heterogeneity increases the improvement of the strategies using all sites over strategies using 2 or 3 sites increases too. Figure 10 shows an example of this fact in a system with 8 sites, 64 processors, $\lambda = 3.0$, and $comm\_comp\_ratio = 0.0025$. This is because small sites generally benefit from using as many remote resources as possible, and using all sites leads to good load balancing.
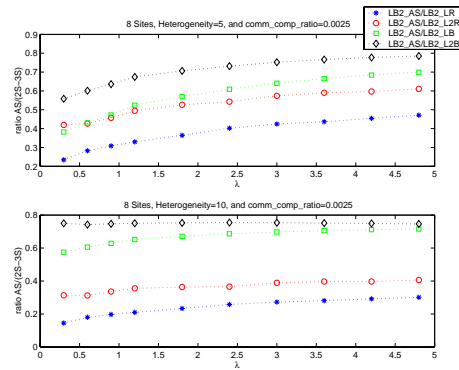

Fig. 9. Comparison between **LB2-AS** and strategies using 2 or 3 sites on a grid platform with 8 sites, 64 total processors, and low heterogeneity (top) and moderate heterogeneity (bottom). $comm\_comp\_ratio = 0.0025$

VI. CONCLUSIONS AND FUTURE WORKS

We evaluated several scheduling approaches on grid platforms for DL jobs, with the goal of answering the questions raised in this paper: (i) *When are the **basic strategies** good enough?*, (ii) *What is the usefulness of global system information for the division policy?*, and (iii) *How many and which remote sites should be picked by the selection policy?*. In support of this investigation, we have developed a discrete-event simulator using the SIMGRID toolkit. We have modeled real-world and synthetic platform as well as real-world and synthetic applications.

Our simulation results lead us to the following conclusions concerning the three questions listed above:

(i) In the presence of applications with high $comm\_comp\_ratio$ and high workload, the simple **B-LS** strategy yields good performance. However, results may be different if the sites have different LS approaches (e.g., some sites not dividing the chunks into
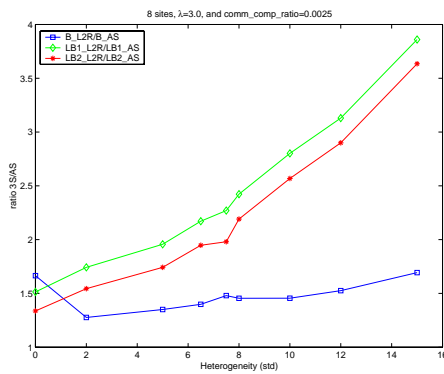
Fig. 10. Comparison between strategies using all sites and strategies using 3 sites on a grid platform with 8 sites, 64 total processors, and $\lambda = 3.0$. $comm\_comp\_ratio = 0.0025$

sub-chunks). The basic strategy using all sites (**B-AS**) reported the worst performance in cases where the total system workload was high.

(ii) Using dynamic FIFO queue sizes information for making scheduling decisions is beneficial when the overall workload is high, with around 10% improvement over using only static site information.

(iii) When heterogeneity increases the improvement of the strategies using all sites over strategies that use 2 or 3 sites increases, but if λ increases this improvement decreases. In systems with few sites and low workload the strategies using 2 sites are better than using all sites, as λ increases using the best sites gets better performance.

We have modeled the workload of jobs arriving to a Poisson process with rate λ. We plan to use other models in future work to see if there is any impact on our simulations.

While a centralized scheduling approach may not be practical, it would be informative to know by how much a distributed approach loses when compared to a centralized one. In future work we will compare our heuristics to the work in [23], which proposes centralized algorithms for scheduling multiple DL applications, in steady state. While their assumptions are different (e.g., jobs are infinite, there are no LSs), a comparison could still be made that sheds light on how much is lost by the decentralized approach.

REFERENCES

[1] D. Altilar and Y. Paker. An Optimal Scheduling Algorithm for Parallel Video Processing. In *Proc. of the IEEE International Conference on Multimedia Computing and Systems*, 1998.

[2] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, and Y. Yang. Scheduling Divisible Loads on Star and Tree Networks: Results and Open Problems. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 16(3):207–218, 2005.

[3] M. A. Bender, S. Chahrabarti, and S. Muthukrishnan. Flow and stretch metrics for scheduling continuous job streams. In *Proc. of the 9th Annual ACM-SIAM Symposium On Discrete Algorithms*, 1998.

[4] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torezon, and R. Wolski. The GrADS Project: Software Support for High-Level Grid Application Development. *International Journal of High Performance Applications and Supercomputing*, 15(4):327–344, 2001.

[5] V. Bharadwaj, D. Ghose, V. Mani, and T. G. Robertazzi. *Scheduling Divisible Loads in Parallel and Distributed Systems*. IEEE Computer Society Press, 1996.

[6] V. Bharadwaj, D. Ghose, and T. G. Robertazzi. Divisible load theory: A new paradigm for load scheduling in distributed systems. *Cluster Computing*, 6(1):7–17, 2003.

[7] A. Bucur and D. Epema. The Maximal Utilization of Processor Co-Allocation in Multicluster Systems. In *Proc. of the International Parallel and Distributed Processing Symposium*, 2003.

[8] A. Bucur and D. Epema. The Performance of Processor Co-Allocation in Multicluster Systems. In *Proc. of the Third IEEE International Symposium on Cluster Computing and the Grid*, 2003.

[9] H. Casanova. Modeling Large-Scale Platforms for the Analysis and the Simulation of Scheduling Strategies. In *Proc. of the 6th Workshop on Advances in Parallel and Distributed Computational Models*, 2004.

[10] T. Davis, A. Chalmers, and H. Wann Jensen. Practical parallel processing for realistic rendering. In *ACM SIGGRAPH*, 2000.

[11] A. Downey and D. Feitelson. The elusive goal of workload characterization. *Performance Evaluation Review*, 26(4):14–29, 1999.

[12] D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn. Parallel job scheduling — a status report. In *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing, Lecture Notes in Computer Science*, volume 3277, pages 1–16, 2004.

[13] NCBI (National Center for Biotechnology Information. Blast. http://www.ncbi.nlm.nih.gov/BLAST/.

[14] J. Gehring and T. Preiss. Scheduling a Metacomputer with Uncooperative Subschedulers. In *Proc. of IPPS Workshop on Job Scheduling Strategies for Parallel Processing*, volume 1659 of *LNCS*, pages 179–201, 1999.

[15] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of Job-Scheduling Strategies for Grid Computing. *LNCS*, 1971:191–202, 2000.

[16] Washington University in St. Louis. Hmmer 2.2. http://hmmer.wustl.edu/hmmer-html.

[17] D. Jackson. MOAB Grid Scheduler (SILVER). http://supercluster.org/projects/silver.

[18] A. Legrand, L. Marchal, and H. Casanova. Scheduling Distributed Applications: The SIMGRID Simulation Framework. In *Proc. of the Third IEEE International Symposium on Cluster Computing and the Grid (CCGrid'03)*, 2003.

[19] B. Lichtenbelt, R. Crane, and S. Naqvi. *Introduction to Volume Rendering*. Prentice Hall PTR, March 1998.

[20] Supercomputer Logs. http://www.cs.huji.ac.il/labs/parallel/workload/.

[21] U. Lublin and D.G. Feitelson. The Workload on Parallel Supercomputers: Modeling the Characteristics of Rigid Jobs. *Journal of Parallel and Distributed Computing*, 63(11):1105–1122, 2003.

[22] Wong H. M., Bharadwaj V., Dantong Y., and T. G. Robertazzi. Data Intensive Grid Scheduling: Multiple Sources with Capacity Constraints. In *Proc. of the International Conference on Parallel and Distributed Computing Systems, (PDCS)*, 2003.

[23] L. Marchal, Y. Yang, H. Casanova, and Y. Robert. A Realistic Network/Application Model for Scheduling Divisible Loads on Large-Scale Platforms. In *Proc. of the International Parallel and Distributed Processing Symposium*, 2005.

[24] MCell Web-page. http://www.mcell.cnl.salk.edu.

[25] T. Moller and E. Haines. *Real-Time Rendering*. A K Peter Ltd, 1st edition, 1999.

[26] K. Ranganathan and I. Foster. Decoupling Computation and Data Scheduling in Distributed Data-intensive Applications. In *Proc. of 11th IEEE International Symposium for High Performance Distributed Computing*, pages 352–358, 2002.

[27] H. Shan, L. Oliker, and R. Biswas. Job Superscheduler Architecture and Performance in Computational Grid Environments. In *Proc. of the 2003 ACM/IEEE conference on Supercomputing*, page 44, 2003.

[28] P. Shirley. *Realistic Ray Tracing*. A K Peters Ltd, 1st edition, June 2000.

[29] V. Subramani, R. Kettimuthu, S. Srinivasan, and P. Sadayappan. Distributed Job Scheduling on Computational Grids using Multiple Simultaneous Requests. In *Proc. of 11th IEEE International Symposium for High Performance Distributed Computing*, pages 359–366, 2002.

[30] TeraGrid Team. TeraGrid Web-page. http://www.teragrid.org.

[31] S. Vadhiyar and J. Dongarra. A Metascheduler for the Grid. In *Proc. of the 11th IEEE Symposium on High-Performance Distributed Computing*, pages 343–351, 2002.