A realistic network/application model for scheduling divisible loads on large-scale platforms

L. Marchal[1], Y. Yang[2], H. Casanova[2,3] and Y. Robert[1]
1: Laboratoire LIP, CNRS-INRIA, École Normale Supérieure de Lyon, France
{Loris.Marchal,Yves.Robert}@ens-lyon.fr
2: Dept. of Computer Science and Engineering 3: San Diego Supercomputer Center
University of California, San Diego, USA
{casanova,yangyang}@cs.ucsd.edu

Abstract

Divisible load applications consist of an amount of data and associated computation that can be divided arbitrarily into any number of independent pieces. This model is a good approximation of many real-world scientific applications, lends itself to a natural master-worker implementation, and has thus received a lot of attention. The issue of divisible load scheduling has been studied extensively. However, only a few authors have explored the simultaneous scheduling of multiple such applications on a distributed computing platform. We focus on this increasingly relevant scenario and make the following contributions. We use a novel and more realistic platform model that captures some of the fundamental network properties of grid platforms. We formulate the steady-state multi-application scheduling problem as a linear program that expresses a notion of fairness between applications. This scheduling problem is NP-complete and we propose several heuristics that we evaluate and compare via extensive simulation experiments. Our main finding is that some of our heuristics can achieve performance close to optimal and we quantify the trade-offs between achieved performance and heuristic complexity.

1. Introduction

A *divisible load* application [11] consists of an amount of computation, or *load*, that can be divided into any number of independent pieces. This corresponds to a perfectly parallel job: any sub-task can itself be processed in parallel, and on any number of workers. The divisible load model is a good approximation for applications that consist of large numbers of identical, low-granularity computations, and has thus been applied to a large spectrum of scientific problems in areas including image processing, volume rendering, bioinformatics, and even data mining. For further information on the model, we refer the reader to [12, 27, 20].

Divisible load applications are amenable to the simple master-worker programming model and can therefore be easily implemented and deployed on computing platforms ranging from small commodity clusters to computational grids. The main challenge is to *schedule* such applications effectively. However, large-scale platforms are not likely to be exploited in a mode dedicated to a single application. Furthermore, a significant portion of the mix of applications running on grid platforms are divisible load applications. At the extreme, a grid such as the CDF Analysis Farms (CAF) [14] supports the concurrent executions of applications that are almost all divisible load applications. Therefore, it is critical to investigate the scheduling of *multiple* divisible loads applications that are executed simultaneously and compete for CPU and network resources.

A first analysis of the concurrent execution of multiple divisible load applications is provided in [10]. The authors target a simple platform composed of a bus network connecting a single master processor to a collection of heterogeneous worker processors. In [30] the authors introduce a (virtual) producer-consumer architecture where several masters are fully connected to a heterogeneous worker processors. The authors describe a strategy for balancing the total amount of work among the workers. Unfortunately, the results are mostly of theoretical interest as it is assumed that masters and workers can communicate with unlimited numbers of concurrent messages, which is unlikely to hold in practice. In [31], the authors discuss how to ap-

This paper is based upon work supported by the National Science Foundation under Grant No. 0234233.

ply divisible load theory to grid computing. They discuss master-worker computation in which the workers are assumed to be only limited by their own network bandwidth and never by internet bandwidth. This assumption does not hold in general.

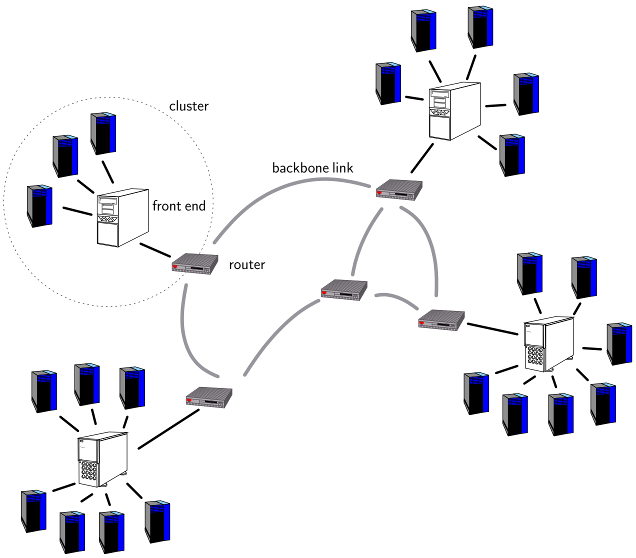

Figure 1. Sample large-scale platform model.

Our contributions are as follows: (i) we propose a new model for deploying and scheduling multiple divisible load applications on large-scale computing platforms, which is significantly more realistic than models used in previous work; (ii) we formulate a relevant multi-application steady-state divisible load scheduling problem, which expresses a notion of fairness among applications, and which is NP-complete; (iii) we propose several polynomial heuristics that we evaluate and compare via extensive simulations. In our model, the target platform consists of a collection of clusters in geographically distributed institutions, interconnected via wide-area networks, as seen in Figure 1. The key benefit of this model is that it takes into account both the inherent hierarchy of the platform and the bandwidth-sharing properties of specific network links. In addition to the new platform model, we adopt a new scheduling objective. Rather than minimizing total application execution time (i.e., the "makespan"), our goal is to maximize the throughput in steady-state mode, i.e., the total load executed per time-period. There are three main reasons for focusing on the steady-state operation. First is *simplicity*, as the steady-state scheduling is really a relaxation of the makespan minimization problem in which the initialization and clean-up phases are ignored. One only needs to determine, for each participating resource, which fraction of time is spent computing for which application, and which fraction of time is spent communicating with which neighbor; the actual schedule then arises naturally from these quantities. Second is *efficiency*, as steady-state scheduling provides, by definition, a periodic schedule, which is described in compact form and is thus possible to implement efficiently in practice. Third is *adaptability*: because the schedule is periodic, it is possible to dynamically record the observed performance during the current period, and to inject this information into the algorithm that will compute the optimal schedule for the next period. This makes it possible to react on the fly to resource availability variations.

2. Platform and Application Model

Our platform model (see Figure 1) consists of a collection of clusters that are geographically distributed over the internet. Each cluster is equipped with a "front-end" processor [11], which is connected to a local router via a local-area link of limited capacity. These routers are used to connect each cluster to the internet. We model the interconnection of all the routers in our platform as a graph of internet backbone links.

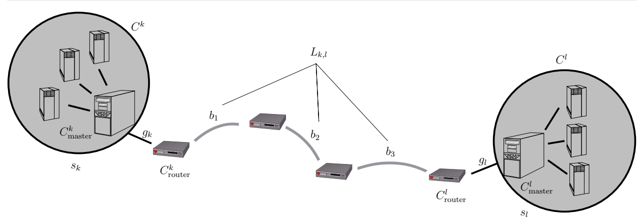

Figure 2. Notations for the platform model.

The inter-cluster graph, denoted as $\mathcal{G}_{ic} = (\mathcal{R}, \mathcal{B})$, is composed of routers (the nodes in $\mathcal{R}$) and backbone links (the edges in $\mathcal{B}$). There are $b = |\mathcal{B}|$ backbone links, $l_1, \ldots, l_b$. For each link we have two parameters: $bw(l_i)$, the bandwidth available for a new connection, and *max-connect*(l_i), the maximum number of connections (in both directions) that can be opened on this link by our applications. The model for the backbones is as follows. Each connection is granted at most a fixed amount of bandwidth equal to $bw(l_i)$, up to the point where a maximum number of connections are simultaneously opened, at which point no more connection can be added. 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 (e.g. in the GridFTP project [1]) and we have observed it in our own measurements [15]. The constraint imposed on the number of allowed connections makes it possible to limit the network usage of applications, which is a likely requirement for future production grid platforms with many applications and users competing for resources.

Compute resources consist of K clusters C^k, $1 \leq k \leq K$. In full generality, we should represent each C^k as a node-weighted edge-weighted graph $G_k = (V_k, E_k)$, but we simplify the model. For each cluster C^k, we only retain C^k_{master}, the front-end processor, which is connected to C^k_{router}, one of the routers in $\mathcal{R}$. The idea is that C^k_{master} represents the cumulated power of the computing resources in the cluster C^k (as shown in Figure 2). This amounts to assuming that the architecture of the cluster is a star-shaped network, whose center is the front-end processor C^k_{master}. It is known that, for the purpose of running divisible load applications, C^k_{master} and the leaf processors are together "equivalent" to a single processor whose speed s_k can be determined as in [26, 4, 2]. In fact, it has also been shown that a tree topology is equivalent to a single processor [4, 3, 5], and thus our model encompasses cases in which the local-area network in each institution is structured as a tree. Consequently, we need only two parameters to characterize each cluster: s_k, the cumulated speed of C^k including C_{master} and the cluster's processors, and g_k, the bandwidth of the link connecting C^k_{master} to C^k_{router}. This link is modeled as follows: any number of connections may share the link, but they each receive a portion of the total available bandwidth, and the sum of these portions cannot exceed g_k, which is known to be a reasonable model for local-area links. Note that this link may correspond to several local area physical links.

Finally, we assume that the routing between clusters is fixed. The routing table contains an ordered list $L_{k,l}$ of backbone links for a connection from cluster C^k to cluster C^l, i.e., from router C^k_{router} to router C^l_{router}. As shown in Figure 2, some intermediate routers may not be associated to any cluster. Also, no specific assumption is made on the interconnection graph. Our model uses realistic bandwidth assignments: we determine the bottleneck link for each end-to-end connection and use the bandwidth-sharing properties of this link (either local-area or backbone) to determine the amount of bandwidth allocated to each connection.

To the best of our knowledge, this model is the first attempt at modeling relatively complex network topologies along with realistic bandwidth-sharing properties for the purpose of large-scale application scheduling research.

3. Steady-State Scheduling of Multiple Applications

The steady-state approach was pioneered by Bertsimas and Gamarnik [8]. Steady-state scheduling allows to relax the scheduling problem The key idea is to characterize the activity of each resource during each time-unit: which (rational) fraction of time is spent computing for which application, which fraction of time is spent receiving or sending to which neighbor. Such activity variables are used to construct a linear program that characterizes the global behavior of the system. Once each activity variable has been computed, the periodic schedule is known: we simply scale the rational values to obtain integer numbers, and the length of the period of the schedule is determined by this scaling.

3.1. Steady-State Equations

We consider K divisible load applications, A_k, $1 \leq k \leq K$, with cluster C^k initially holding all the input data necessary for application A_k. For each application we define a "priority factor", π_k, that quantifies its relative worth. For instance, computing two units of load per time unit for an application with priority factor 2 is as worthwhile/profitable than computing one unit of load for an application with priority factor 1. This concept makes it possible to implement notions of application priorities for resource sharing. We can easily refine the priority model and define $\pi_{k,l}$ as the priority factor to execute a fraction of application A_k onto cluster C^l. Similarly, our method is easily extensible to the case in which more than one application originates from the same cluster. We start with the following three definitions:

w_k **and** δ_k **(load unit size for** A_k**)** – The divisible applications may be of different types. For instance one application may deal with files and another with matrices. We divide each application into load units (a file, or a matrix). We let w_k be the amount of computation required to process a load unit for application A_k. Similarly, δ_k

is the size (in bytes) of a load unit for application A_k.

$\alpha_{k,l}$ **(fraction of A_k executed by C^l)** – Each cluster C^k initially holds input data for application A_k. Within a time-unit, C^k will devote a fraction of the time to process load units for application A_k. But cluster C^k can also be used to process loads that originates from another cluster C^l, i.e., from application A_l. Reciprocally, portions of application A_k may be executed by other clusters. We let $\alpha_{k,l}$ be the portion of load for application A_k that is sent by C^k and computed on cluster C^l within a time-unit. $\alpha_{k,k}$ denotes the portion of application A_k which is executed on the local cluster.

$\beta_{k,l}$ **(connections from C^k to C^l)** – Cluster C^k opens $\beta_{k,l}$ network connections to send the portion $\alpha_{k,l}$ of application A_k that is destined to cluster C^l.

With the above definitions, it takes $\frac{\alpha_{k,l}.w_k}{s_l}$ time-units to process $\alpha_{k,l}$ load units of application A_k on cluster C^l. Similarly, it takes $\frac{\alpha_{k,l}.\delta_k}{g_{k,l}}$ time-units to send $\alpha_{k,l}$ load units of application A_k along a single network connection from router C^k_{router} to router C^l_{router}, where $g_{k,l}$ is the minimum bandwidth available for one connection on a route from cluster C^k to cluster C^l, i.e. $g_{k,l} = \min_{l_i \in L_{k,l}} \{bw(l_i)\}$.

The first steady-state equation states that a cluster C^k cannot compute more load units per time unit than allowed by its speed s_k:

$$\forall C^k, \quad \sum_l \alpha_{l,k} \cdot w_l \quad \leq \quad s_k \tag{1}$$

With steady-state scheduling we do not need to determine the precise ordering in which the different load types are executed by C^k: instead we take a macroscopic point of view and simply bound the total amount of load processed every time-unit.

The second steady-state equation bounds the amount of load that requires the use of the serial link between cluster C^k and the external world, i.e., between C^k_{master} and C^k_{router}:

$$\forall C^k, \underbrace{\sum_{l \neq k} \alpha_{k,l} \cdot \delta_k}_{\text{(outgoing data)}} + \underbrace{\sum_{j \neq k} \alpha_{j,k} \cdot \delta_j}_{\text{(incoming data)}} \quad \leq \quad g_k \tag{2}$$

This equation states that the available bandwidth g_k is not exceeded by the requirements of all the traffic outgoing from and incoming to cluster C^k. Note that we assume that the time to execute a portion of an application's load, or to communicate it along a serial link, is proportional to its size in number of load units: this amounts to fixing the granularity and to manipulating load units. Start-up costs could be included in the formulas, but at the price of technical difficulties: only asymptotic performance can be assessed in that case [6].

Next we must bound the utilization of the backbone links. Our third equation states that on each backbone link l_i, there should be no more than $\textit{max-connect}(l_i)$ opened connections:

$$\sum_{\{k,l\},\ l_i \in L_{k,l}} \beta_{k,l} \quad \leq \quad \textit{max-connect}(l_i) \tag{3}$$

The fourth equation states that there is enough bandwidth on each path from a cluster C^k to a cluster C^l:

$$\alpha_{k,l} \cdot \delta_k \quad \leq \quad \beta_{k,l} \times g_{k,l}. \tag{4}$$

The last term $g_{k,l}$ in Equation 4 was defined earlier as the bandwidth allotted to a connection from C^k to C^l. This bandwidth is simply the minimum of the $bw(l_i)$, taken over all links l_i on the path from C^k to C^l. We multiply this bandwidth by the number of opened connections to derive the constraint on $\alpha_{k,l}$.

Finally there remains to define an optimization criterion. Let $\alpha_k = \sum_{l=1}^{K} \alpha_{k,l}$ be the load processed for application A_k per time unit. To achieve a fair balance of resource allocations one could execute the same number of load units per application, and try to maximize this number. However, some applications may have higher priorities than others, hence the introduction of the priority factors π_k in the objective function:

$$\textsc{Maximize} \quad \min_k \left\{ \frac{\alpha_k}{\pi_k} \right\}. \tag{5}$$

This maximization corresponds to the well-known MAX-MIN fairness strategy [7] between the different loads, with coefficients $1/\pi_k$, $1 \leq k \leq K$. The constraints and the objective function form a linear program:

$$\begin{array}{l}
\textsc{Maximize } \min_k \left\{ \frac{\alpha_k}{\pi_k} \right\}, \\
\textsc{under the constraints} \\
\left\{ \begin{array}{lll}
\text{(6a)} & \forall C^k, & \sum_l \alpha_{k,l} = \alpha_k \\
\text{(6b)} & \forall C^k, & \sum_l \alpha_{l,k} \cdot w_l \leq s_k \\
\text{(6c)} & \forall C^k, & \sum_{l \neq k} \alpha_{k,l} \cdot \delta_k + \sum_{j \neq k} \alpha_{j,k} \cdot \delta_j \leq g_k \\
\text{(6d)} & \forall i, & \sum_{L_{k,l} \ni l_i} \beta_{k,l} \leq \textit{max-connect}(l_i) \\
\text{(6e)} & \forall k,l, & \alpha_{k,l} \cdot \delta_k \leq \beta_{k,l} \cdot g_{k,l} \\
\text{(6f)} & \forall k,l, & \alpha_{k,l} \geq 0 \\
\text{(6g)} & \forall k,l, & \beta_{k,l} \in \mathbb{N}
\end{array} \right.
\end{array} \tag{6}$$

This program is mixed as the $\alpha_{k,l}$ are rational numbers but the $\beta_{k,l}$ are integers. Given a platform $\mathcal{P}$ and computational priorities $(\pi_1, \ldots, \pi_K)$, we define a *valid allocation* for the steady-state mode as a set of values (α, β) such that Equations (6) are satisfied. Since this program involves integer variables there is little hope that an optimal solution could be computed in polynomial time. It turns out that this is an NP-hard problem, and we refer the reader to a technical report for a formal proof [25].

3.2. Reconstructing a Periodic Schedule

Once one has obtained a solution to the linear program defined in the previous section, say (α, β), one needs to reconstruct a (periodic) schedule, that is a way to decide in which specific activities each computation and communication resource is involved during each period. This is straightforward because the divisible load applications are independent of each other. We express all the rational numbers $\alpha_{k,l}$ as $\alpha_{k,l} = \frac{u_{k,l}}{v_{k,l}}$, where the $u_{k,l}$ and the $v_{k,l}$ are relatively prime integers. The period of the schedule is set to $T_p = \mathbf{lcm}_{k,l}(v_{k,l})$. In steady-state, during each period of length T_p:

- Cluster C^k computes, for each non-zero value of $\alpha_{l,k}$, $\alpha_{l,k} \cdot T_p$ load units of application A_l. If $l = k$ the data is local, and if $k \neq l$, the data corresponding to this load has been received during the previous period. These computations are executed in any order. Equation 1 ensures that $\frac{\sum_l \alpha_{l,k} \cdot w_l \cdot T_p}{T_p} \leq s_k$, hence C^k can process all its load.
- Cluster C^k sends, for each non-zero value of $\alpha_{k,l}$, $\alpha_{k,l} \cdot \delta_k \cdot T_p$ load units of application A_k, to be processed by cluster C^l during the next period. Similarly, it receives, for each non-zero value of $\alpha_{j,k}$, $\alpha_{j,k} \cdot zc_j \cdot T_p$ load units for application A_j, to be processed locally during the next period. All these communications share the serial link, but Equation 2 ensures that $\frac{\sum_{l \neq k} \alpha_{k,l} \cdot \delta_k \cdot T_p + \sum_{j \neq k} \alpha_{j,k} \cdot \delta_j \cdot T_p}{T_p} \leq g_k$, hence the link bandwidth is not exceeded.

Obviously, the first and last period are different: no computation takes place during the first period, and no communication during the last one. Altogether, we have a periodic schedule, which is described in compact form: we have a polynomial number of intervals during which each processor is assigned a given load for a prescribed application.

4. Heuristics

We propose several heuristics to solve our scheduling problem. We first propose a greedy heuristic, and then heuristics that are based on the rational solution to the mixed linear program derived in Section 3.

4.1. Greedy Heuristic

Our greedy heuristic, which we simply call G, allocates resources to one of the K applications in a sequence of steps. More specifically, at each step the heuristic (i) selects an application A_k; (ii) determines on which cluster C^l the work will be executed (locally if $l = k$, on some remote cluster otherwise); and (iii) decides how much work to execute for this application. The intuition for how these choices can be made is as follows:

- One should select the application that has received the smallest relative share of the resource so far, that is the one for which α_k / π_k is minimum, where $\alpha_k = \sum_l \alpha_{k,l}$. Initially, $\alpha_k = 0$ for all k, so one can break ties by giving priority to the application with the highest priority factor π_k.
- Compare the payoff of computing on the local cluster with the payoff of opening one connection to each remote cluster. Choose the most profitable cluster, say C^l.
- Allocate an amount of work that does not overload C^l so that it will not be usable by other applications.

The greedy heuristic, which we denote by G, is formalized as follows:

1. Let $L = \{C^1, \ldots, C^K\}$. Initialize all $\alpha_{k,l}$ and $\beta_{k,l}$ to 0.
2. If L is empty, exit.
3. **Select application** – Sort L by non-decreasing values of $\left(\frac{\alpha_k}{\pi_k}\right)$. Break ties by choosing the application with larger priority first. Let k be the index of the first element of L. Select A_k.
4. **Select cluster** – For each cluster C^m where $m \neq k$, compute the work (i.e., number of load units for A_k) that can be executed using a single connection: $\text{benefit}_m = \min\left\{\frac{g_k}{\delta_k}, \frac{g_{k,m}}{\delta_k}, \frac{g_m}{\delta_k}, \frac{s_m}{w_k}\right\}$. Locally, one can achieve $\text{benefit}_k = \frac{s_k}{w_k}$. Select C^l, $1 \leq l \leq K$ so that benefit_l is maximal. If $\text{benefit}_l = 0$ (i.e., no more work can be executed), then remove C^k from list L and go to step 2.
5. **Determine amount of work** – If $k \neq l$ (remote computation), allocate $\text{alloc} = \text{benefit}_l$ units of load to cluster C^l. If $k = l$ (local computation), allocate only

alloc $= \max_{m \neq k} \left\{ \min \left\{ \frac{g_k}{\delta_k}, \frac{g_{k,m}}{\delta_k}, \frac{g_m}{\delta_k}, \frac{s_m}{w_k} \right\} \right\}$ units of load. This quantity is the largest amount that could have been executed on C^k for another application and is used to prevent over-utilization of the local cluster early on in the scheduling process.

6. **Update variables** –
   - Decrement speed of target cluster C^l: $s_l \leftarrow s_l - \text{alloc}.w_k$
   - Allocate work: $\alpha_{k,l} \leftarrow \alpha_{k,l} + \text{alloc}$
   - In case of a remote computation (if $k \neq l$) update network characteristics:

$$\forall l_i \in L_{k,l}, \quad \textit{max-connect}(l_i) \leftarrow \textit{max-connect}(l_i) - 1$$
$$g_k \leftarrow g_k - \text{alloc}.\delta_k, \quad g_l \leftarrow g_l - \text{alloc}.\delta_k,$$
$$\beta_{k,l} \leftarrow \beta_{k,l} + 1$$

7. Go to step 2.

4.2. LP-Based Heuristics

The linear program given in Section 3 is a mixed integer/rational numbers linear program since the variables $\beta_{k,l}$ take integer values and variables $\alpha_{k,l}$ may be rational. This mixed LP (MLP) formulation gives an exact optimal solution to the scheduling problem, while a rational LP formulation allows rational $\beta_{k,l}$ and gives an *upper bound* of the optimal solution. As solving a mixed linear program is known to be hard, we propose several heuristics based on the relaxation of the problem: we first solve the linear program over the rational numbers with a standard method (e.g., the Simplex algorithm). We then try to derive a solution with integer $\beta_{k,l}$ from the rational solution.

4.2.1. LPR: Round-off The most straightforward approach is to simply round rational $\beta_{k,l}$ values to the largest smaller integer. Formally, if $(\widetilde{\alpha}_{k,l}, \widetilde{\beta}_{k,l})$ is a rational solution to the linear program, we build the following solution:

$$\forall k,l, \quad \widehat{\beta}_{k,l} = \lfloor \widetilde{\beta}_{k,l} \rfloor, \quad \widehat{\alpha}_{k,l} = \min \left\{ \widetilde{\alpha}_{k,l}, \frac{\lfloor \widetilde{\beta}_{k,l} \rfloor \cdot g_{k,l}}{\delta_k} \right\}.$$

With these new values, we have $\widehat{\beta}_{k,l} \leq \widetilde{\beta}_{k,l}$ and $\widehat{\alpha}_{k,l} \leq \widetilde{\alpha}_{k,l}$ for all indices k,l. Furthermore, $(\widehat{\alpha}, \widehat{\beta})$ is a valid solution to the mixed linear program (6) in which all $\widehat{\beta}_{k,l}$ take integer values. We label this method LPR.

4.2.2. LPRG: Round-off + Greedy Rounding down all the $\beta_{k,l}$ variables with LPR may lead to a very poor result as the remaining network capacity is unutilized. The LPRG heuristic reclaims this residual capacity by applying the technique described in Section 4.1. Intuitively, LPR gives the basic framework of the solution, while the Greedy heuristic refines it.

4.2.3. LPRR: Randomized Round-off Relaxing an integer linear program into rational numbers is a classical approach, and several solutions have been proposed. Among them is the use of randomized approximation. In [21, chapter 11] Motwani, Naor and Raghavan propose this approach to solve a related problem, the multicommodity flow problem. Using Chernoff bounds, they prove that their algorithm leads, with high probability, to a feasible solution that achieves the optimal throughput. Although this theoretical result seems attractive, it has some drawbacks for our purpose. First, our problem is not a multicommodity flow problem: instead of specifying a set of flow capacities for between node pairs, we have global demands for the sum of all flows leaving each node (representing the total amount of work sent by this node). Second, to obtain their optimality result, the authors in [21, chapter 11] rely on the assumption that the capacity of each edge is not smaller than a bound ($5.2 \times \ln(4m)$ where m is the number of edges), and we do not have a similar property here. Third, there are two cases of failure in the randomized algorithm (even though the probability of such failures is proved to be small): either the algorithm provides a solution whose objective function is suboptimal (which is acceptable), or it provides a solution which does not satisfy all the constraints (which is not acceptable).

Coudert and Rivano proposed in [18] a rounding heuristic based on the method of [21, chapter 11] in the context of optical networks. Their method seems more practical as it always provides a feasible solution. We use a similar approach and our heuristic, LPRR, works as follows:

1. Solve the original linear program with rational numbers. Let $(\widetilde{\alpha}_{k,l}, \widetilde{\beta}_{k,l})$ be the solution.
2. Choose a route k,l at random, such that $\widetilde{\beta}_{k,l} \neq 0$.
3. Randomly choose $X_{k,l} \in \{0,1\}$ with probability $P(X_{k,l} = 1) = \widetilde{\beta}_{k,l} - \lfloor \widetilde{\beta}_{k,l} \rfloor$.
4. Assign the value $v = \lfloor \widetilde{\beta}_{k,l} \rfloor + X$ to $\beta_{k,l}$ by adding the constraint $\beta_{k,l} = v$ to the linear program.
5. If there is at least a route k,l for which no $\beta_{k,l}$ value has been assigned, go to step 2.

Note that LPRR solves K^2 linear programs, and is thus much more computationally expensive that our other LP-based heuristics.

5. Experimental Results

5.1. Methodology

In this section, we use simulation to evaluate the G, LPR, LPRG, and LPRR heuristics. Ideally the objective values achieved by these heuristics should be compared to the optimal solution, i.e., the solution to the mixed linear problem. However, solving the mixed linear problem takes exponential time and we cannot compute its solution in practice. Instead we use the solution to the *rational* linear problem as a comparator, as it provides an upper bound on the optimal solution (i.e., it cannot be achieved/used in practice as $\beta_{k,l}$ values need to be integers).

One important question for creating relevant instances of our problem is that of the network topology. We opted for using the popular Tiers [22] topology generator to create a comprehensive set of topologies. We randomly generated 100 two-level topologies with Tiers, each topology containing 40 WAN nodes, 30 MAN networks each containing 20 MAN nodes. We did not generate LAN networks as in our model we abstract them as a single cluster/site that delivers computation to the applications. We set a high connection redundancy value to reflect the rich connectivity between backbone nodes. Each of these topologies contains approximately 700 nodes. For each topology, we randomly select $K = 5, 7, \ldots, 90$ nodes as clusters participating in the computation of divisible load applications. For these K nodes we determine all pair-wise shortest paths (in hops), and we then delete the nodes not on any shortest paths, so as to be left with the topology interconnecting the sites participating in computation. Note that in this "pruned" topology, there are nodes that we did not originally select but happen to be on the shortest paths between nodes that we had selected. We consider these nodes purely as routers that do not perform any computation, which can be easily expressed in the linear program defined in Section 3.1 by adding corresponding constraints but not modifying the objective function.

Now that the topology is specified, we assign ranges of values to s_k, g_k, max-connect(l_k), δ_k, w_k, and π_k as follows. The local bandwidth at each site, g_k, and the link bandwidth, $bw(l_i)$, is set according to a comprehensive measurement of internet end-to-end bandwidths [23]. This study shows that the logarithm of observed data transfer rates are approximately normally distributed with mean log(2000kbits/sec), and standard deviation log(10), which we use to generate random values for g_k and $bw(l_i)$. We generate all the other parameters according to uniform distributions with ranges shown in Table 1. For each of our pruned Tiers topologies we generate 10 platform configurations with random instantiations of the above parameters. In total, we perform experiments over 29,298 generated platform configurations.

parameter	distribution
K	$5, 7, \ldots, 90$
$\log(bw(l_k))$, $\log(g_k)$	normal (*mean*= log(2000), *std*=log(10))
s_k	uniform, 1000 — 10000
max-connect, δ_k, w_k, π_k	uniform, 1 — 10

Table 1. Platform parameters used in simulation

5.2. Results

LPR – Our first (expected) observation from our simulation results is that LPR always performs poorly. In most cases, LPR leaves a significant portion of the network capacity unutilized, and in some cases all $\beta_{k,l}$ values are actually rounded down to 0, leading to an objective value of 0.

G v.s. LPRG – More interesting is the comparison between G and LPRG. Unlike what we initially expected, G performs consistently better than LPRG. Over all platform configurations, the average ratio of the objective values achieved by G to that by LPRG is 1.18, with a standard deviation of 31.5, and G is better than LPRG in 81% of the cases. For a closer look, Figure 3 plots the average ratio of the objective values achieved by G and LPRG to the upper bound of the optimal obtained by solving the rational linear program, versus the number of clusters K. We see that G achieves objective values about 5% $\sim$ 10% higher than LPRG in most cases. But as K increases, both heuristics fail to achieve objective values close to the upper bound of the optimal.

To explain the poor performance of LPRG relatively to G, we examined the simulation logs closely. It turns out that, after solving the rational linear program, there often are some clusters that send a portion of their load to other clusters using a rational number of network connections that is strictly lower than 1. After rounding this value down to 0, such clusters have then no opportunity to send off this load portion and become oversubscribed: they take the objective value of the rational linear program down. Conversely, the clusters that were supposed to receive this load are now undersubscribed and have cycles to spare. During the greedy step, the G heuristic sometimes picks

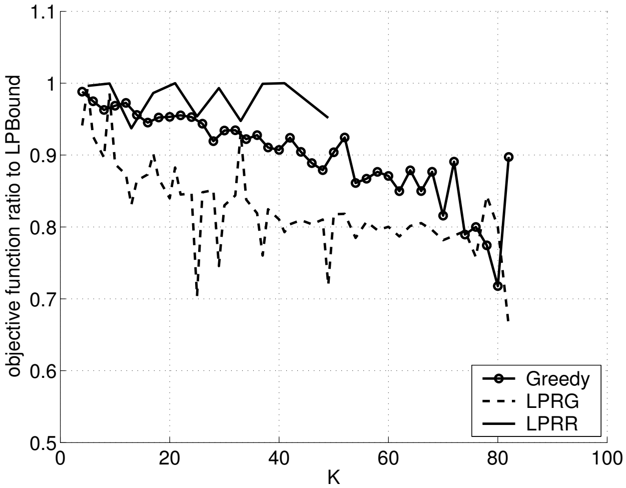

Figure 3. Performance of G, LPRG and LPRR, relative to the upper bound of the optimal.

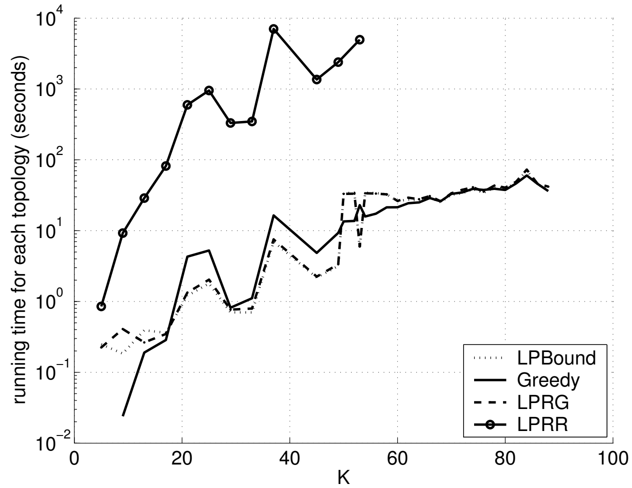

Figure 4. Running time of G,LPRG and LPRR

an undersubscribed cluster first, which causes this cluster to use up its own spare cycles for its own load. This does not help the MAX-MIN objective value as this cluster was typically better off than the oversubscribed clusters. Furthermore, one or more oversubscribed clusters have now lost the opportunity to use these cycles, which harms the MAX-MIN objective value. By contrast, when the G heuristic starts from scratch, it balances the load better by allowing such undersubscribed clusters to use a full network connection early on in the resource allocation process.

It is interesting to note that the relative performance of G and LPRG is dependent on the type of topology. Indeed, we ran simulations over topologies created as simple random graphs (i.e., any two nodes are connected with a fixed probability). Over these topologies, LPRG heuristic achieved, on average, better results than G. While these random topologies are not representative of actual networks, it would be interesting to understand which properties of the interconnection topology affect the relative performance of G and LPRG.

LPRR – Figure 3 also shows that LPRR performs consistently better than both G and LPRG, and in fact, LPRR achieves an objective value quite close to the upper bound of the optimal, even at $K = 45$. Because LPRR is much more time consuming than the other heuristics, taking K^2 time, we evaluated LPRR only on 62 topologies, and limited the topologies to $K < 50$.

Running Time – Figure 4 shows how time-consuming each of our heuristics are and plots their running time in seconds on a 1GHz Pentium processor versus the number of clusters, on a logarithmic scale. We see that LPRR is very expensive: at $K = 50$, each run of LPRR takes approximately 1 hour. This implies that for implementing a scheduler on a real platform with many clusters G and LPRG are more practical candidates than LPRR.

6. Perspectives on Implementation

Our work has investigated strategies for steady-state scheduling of multiple divisible load applications in a grid environment. In this section we discuss how our work and results could be implemented as part of a framework for deploying divisible load applications. Consider a Virtual Organization (VO) [19] in which participating sites hold resources that they are willing to contribute for the execution of divisible load applications, as well as users who wish to execute such applications. The G heuristic could be implemented as part of a centralized broker that would manage divisible load applications and the resources they can use, for the entire VO. VO participants would register their resources to the broker, and application requests would be submitted to the broker by users. Note that because our work aims at optimizing steady-state throughput, it provides very good schedules for situations in which applications run for a significant amount of time so that the start-up and clean-up phases of application executions are negligible when compared to the entire application execution time. This is a likely scenario in a VO that supports VO-wide application executions.

The broker needs to gather all relevant information to instantiate the LP formulation of the scheduling problem (which is needed to implement the G heuristic as well), as given in Section 3.1. Most important are the π_k coefficients that are used to define the objective function. These coefficients define the policies that govern resource sharing over the entire grid, and these policies should be configured at the broker by a VO administrator (or by any kind of contracting system that is in place in the VO). As mentioned in Section 3.1, the objective function can be extended so that different weights are associated for each pair of sites, $\pi_{k,l}$, thereby quantifying peering relationship between VO participants. Furthermore, other constraints can be added to the linear program to reflect other arbitrary resource sharing policies (e.g., no more than 10% of resources at cluster k can be used for applications originating from clusters i and j). The main point here is that our linear program can be refined to express a wide variety of resource sharing policies and peering relationships among VO participants. It would be interesting to see how the G heuristic compares to LPRG and LPRR for more constrained scheduling problems. The broker also needs to be configured so that the number of network connections used by divisible load applications in the VO does not exceed the max-connect threshold. While in this paper we have looked at a general model in which every link has its own threshold, in practice the VO administrator may not have sufficient knowledge of the network topology. In this case VO administrators would just configure a limit on the number of connections on each path between each pair of participating sites, and the same limit could be enforced for each path. The broker needs to have estimates of the compute and transfer speeds that are achieved on the resources. This can be done by querying grid information services, or by directly observing the performance being delivered by the resources. The later method may prove easier to implement if divisible load applications are continuously running. The best solution is probably to use both methods and combine them to obtain estimates of achievable performance, as done for instance in [16]. The broker needs to adapt its scheduling decisions as resource availability fluctuates and as applications start and complete. One simple option is to allow adaptation to occur after each scheduling period. Additionally, the schedule could be recomputed on-the-fly as soon as a new application is submitted to the broker or a running application completes.

7. Conclusion

We have addressed the steady-state scheduling problem for multiple concurrent divisible applications running on platforms that span multiple clusters distributed over wide-area networks. This is an important problem as divisible load applications are common and make up a significant portion of the mix of grid applications. Only a few authors had explored the simultaneous scheduling of multiple such applications on a distributed computing platform [10, 30] and in this paper we have made the following contributions. We defined a realistic platform model that captures some of the fundamental network properties of grid platforms. We then formulated our scheduling problem as a mixed integer-rational linear program that enforces a notion of weighted priorities and fairness for resource sharing between applications. We proposed a greedy heuristic, G, and three heuristics based on the rational solution to the linear program: LPR, LPRG, and LPRR. We evaluated these heuristics with extensive simulation experiments for many random platform configurations whose network topologies were generated by Tiers [22]. We found that the G heuristic performs better than LPRG on average, and that its performance relative to an upper bound of the optimal decreases with the number of clusters in the platform. We also found that the LPRR heuristic leads to better schedules than G but at the cost of a much higher complexity, which may make it impractical for large numbers of clusters.

We will extend this work in several directions. First, we will simulate platforms and application parameters that are measured from real-world testbeds and applications suites [9, 28]. While this paper provides convincing evidence about the relative merit of our different approaches, simulations instantiated specifically with real-world data will provide a quantitative measure of absolute performance levels that can be expected with the best heuristics. Second, we will strive to use an even more realistic network model, which would include link latencies, TCP bandwidth sharing behaviors according to round-trip times, and more precise backbone characteristics. Some of our recent work (see [24, 15]) provides the foundation for refining our network model, both based on empirical measurements and on theoretical modeling of network traffic.

References

[1] W. Allcock, J. Bester, J. Bresnahan, A. Chervenak, L. Liming, and S. Tuecke. GridFTP: Protocol Extension to FTP for the Grid. Grid Forum Internet-Draft, March 2001.

[2] C. Banino, O. Beaumont, L. Carter, J. Ferrante, A. Legrand, and Y. Robert. Scheduling strategies for master-slave tasking on heterogeneous processor

platforms. *IEEE Trans. Parallel Distributed Systems*, 15(4):319–330, 2004.

[3] G. Barlas. Collection-aware optimum sequencing of operations and closed-form solutions for the distribution of a divisible load on arbitrary processor trees. *IEEE Trans. Parallel Distributed Systems*, 9(5):429–441, 1998.

[4] S. Bataineh, T. Hsiung, and T.G.Robertazzi. Closed form solutions for bus and tree networks of processors load sharing a divisible job. *IEEE Transactions on Computers*, 43(10):1184–1196, Oct. 1994.

[5] O. Beaumont, H. Casanova, A. Legrand, Y. Robert, and Y. Yang. Scheduling divisible loads for star and tree networks: main results and open problems. Technical Report RR-2003-41, LIP, ENS Lyon, France, Sept. 2003. To appear in *IEEE Trans. Parallel and Distributed Systems*.

[6] O. Beaumont, A. Legrand, and Y. Robert. Scheduling divisible workloads on heterogeneous platforms. *Parallel Computing*, 29:1121–1152, 2003.

[7] D. Bertsekas and R. Gallager. *Data Networks*. Prentice Hall, 1987.

[8] D. Bertsimas and D. Gamarnik. Asymptotically optimal algorithm for job shop scheduling and packet routing. *Journal of Algorithms*, 33(2):296–318, 1999.

[9] M. D. Beynon, T. Kurc, A. Sussman, and J. Saltz. Optimizing execution of component-based applications using group instances. *Future Generation Computer Systems*, 18(4):435–448, 2002.

[10] V. Bharadwaj and G. Barlas. Efficient scheduling strategies for processing multiple divisible loads on bus networks. *Journal of Parallel and Distributed Computing*, 62:132–151, 2002.

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

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

[13] T. Braun, H. Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. Reuther, J. P. Robertson, M. Theys, B. Yao, D. Hensgen, and R. F. Freund. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, 2001.

[14] CDF Analysis Farms (CAF). `http://cdfcaf.fnal.gov/`.

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

[16] H. Casanova and F. Berman. *Grid Computing: Making The Global Infrastructure a Reality*, chapter Parameter Sweeps on the Grid with APST. John Wiley, 2003. Hey, A. and Berman, F. and Fox, G., editors.

[17] S. Chakrabarti, J. Demmel, and K. Yelick. Models and scheduling algorithms for mixed data and task parallel programs. *Journal of Parallel and Distributed Computing*, 47:168–184, 1997.

[18] D. Coudert and H. Rivano. Lightpath assignment for multifibers WDM optical networks with wavelength translators. In *IEEE Global Telecommunications Conference (Globecom'02)*. IEEE Computer Society Press, 2002. Session OPNT-01-5.

[19] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of High Performance Computing Applications*, 15(3), 2001.

[20] D. Ghose and T. Robertazzi, editors. *Special issue on* Divisible Load Scheduling. Cluster Computing, 6, 1, 2003.

[21] D. Hochbaum. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, 1997.

[22] K. Calvert and M. Doar and E.W. Zegura. Modeling Internet Topology. *IEEE Communications Magazine*, 35:160–163, 1997.

[23] C. Lee and J.Stepanek. On Future Global Grid Communications Performance. In *HCW'2001, the 10th Heterogeneous Computing Workshop*. IEEE Computer Society Press, 2001.

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

[25] L. Marchal, Y. Yang, H. Casanova, and Y. Robert. A realistic network/application model for scheduling divisible loads on large-scale platforms. Research Report RR-2004-21, LIP, ENS Lyon, France, Apr. 2004.

[26] T. Robertazzi. Processor equivalence for a linear daisy chain of load sharing processors. *IEEE Trans. Aerospace and Electronic Systems*, 29:1216–1221, 1993.

[27] T. Robertazzi. Ten reasons to use divisible load theory. *IEEE Computer*, 36(5):63–68, 2003.

[28] M. Spencer, R. Ferreira, M. Beynon, T. Kurc, U. Catalyurek, A. Sussman, and J. Saltz. Executing multiple pipelined data analysis operations in the grid. In *2002 ACM/IEEE Supercomputing Conference*. ACM Press, 2002.

[29] J. Subhlok and G. Vondran. Optimal use of mixed task and data parallelism for pipelined computations. *Journal of Parallel and Distributed Computing*, 60:297–319, 2000.

[30] H. Wong, D. Yu, , V. Bharadwaj, and T. Robertazzi. Data intensive grid scheduling: multiple sources with capacity constraints. In *PDCS'2003, 15th Int'l Conf. Parallel and Distributed Computing and Systems*. IASTED Press, 2003.

[31] D. Yu and T. Robertazzi. Divisible load scheduling for grid computing. In *PDCS'2003, 15th Int'l Conf. Parallel and Distributed Computing and Systems*. IASTED Press, 2003.