Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

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

Loris Marchal,
Yang Yang,
Henri Casanova,
Yves Robert

April 2004

Research Report N° 2004-21

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

INRIA

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

Loris Marchal, Yang Yang, Henri Casanova, Yves Robert

April 2004

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 critical issue of divisible load scheduling has been studied extensively in previous work. 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 a steady-state multi-application scheduling problem as a linear program that expresses some 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 conducted over 250,000 platform configurations. Our main finding is that some of our heuristics can achieve performance close to the optimal and we quantify the trade-offs between achieved performance and heuristic complexity.

Keywords: parallel computing, scheduling, divisible load, bandwidth sharing, resource sharing, multiple applications.

Résumé

Le modèle des tâches divisibles s'applique aux applications dont les données (et les calculs associés) ont une granularité arbitraire. Ce modèle a fait l'objet de nombreuses études, car son champ d'application est vaste, et sa mise en oeuvre facile, via un paradigme maître-esclave classique. Nous étudions ici le déploiement simultané de *plusieurs* applications divisibles. Nous introduisons un modèle de plate-forme bien plus réaliste que les modèles précédents, et plus proche des caractéristiques des architectures distribuées à grande échelle. Nous cherchons à optimiser le régime permanent, avec un objectif qui prend en compte un accès équitable aux ressources pour les différentes applications. Ce problème d'ordonnancement est NP-complet, aussi nous proposons plusieurs heuristiques. Nous évaluons et comparons celles-ci à travers une large gamme de simulations expérimentales conduites sur plus de 250 000 configurations. Certaines heuristiques sont très proches de l'optimal, et nous quantifions les compromis nécessaires entre performance et temps de calcul.

Mots-clés: parallélisme, ordonnancement, modèle des tâches divisibles, partage de bande passante, partage de ressources, multi-applications.

1 Introduction

A *divisible load* application [15] 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. For further information on the model, we refer the reader to the recent surveys [16, 31], to the special issue of the Cluster Computing journal [25], and to the Web page collecting related literature [29].

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 [22]. 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) [18] supports the concurrent executions of applications that are almost all divisible load applications. Therefore, it is critical to investigate scenarios in which *multiple* divisible loads applications are simultaneously executed on the platform, and therefore compete for CPU and network resources.

A first analysis of the concurrent execution of multiple divisible load applications is provided in [14]. The authors target a simple platform composed of a bus network connecting the master processor (which initially holds the entire load to be distributed and computed) to a collection of heterogeneous, i.e. different-speed, worker processors. Both single-round and multiple rounds strategies are introduced and analyzed.

A more complex platform has been investigated in [34]. In this paper, the authors introduce a (virtual) producer-consumer architecture where a number of data servers (the sources of the multiple divisible loads) are fully connected to a number of heterogeneous processors (the workers that execute the multiple loads). The authors describe a strategy to balance the total amount of work among the workers. Unfortunately, the results are mostly of theoretical interest as the authors assume that a data server can emit an unlimited number of messages in parallel and, similarly, that a computing processor can simultaneously receive an arbitrary number of messages. Obviously, these assumptions are not likely to hold on real-world platforms.

In the position paper [35], the authors discuss how to apply divisible load theory to grid computing [23, 10]. They discuss job scheduling policies for a master-worker computation where the master distributes jobs to remote workers. The latter are assumed to be only limited by their own network bandwidth, and never by internet bandwidth. As a result, the architecture behaves like a simple star-shaped network with single-link connections, and ignores the fact that workers are actually located within geographically dispersed sites.

In this paper we make the following contributions: (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 and show that it 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. We detail and justify our model further in Section 2.

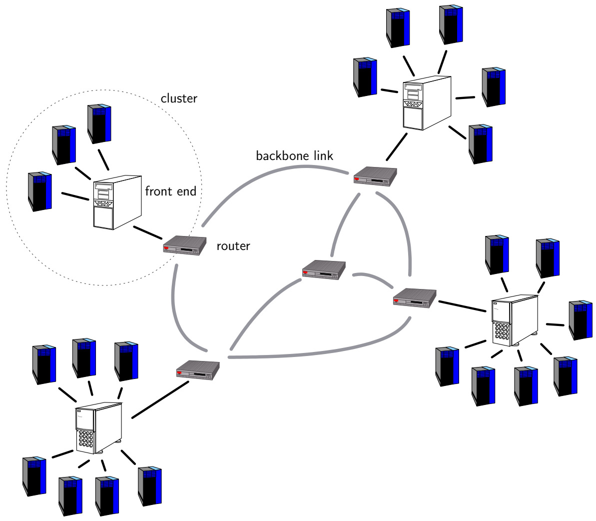

Figure 1: Sample large-scale platform model.

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:

(i) Simplicity: Steady-state scheduling can be viewed as a relaxation of the makespan minimization problem in which the initialization and clean-up phases are ignored. One only needs to determine which fraction of time each participating resource spends computing for which application and spends communicating with which neighbor; the actual schedule then arises naturally from these quantities.

(ii) Efficiency: Steady-state scheduling provides, by definition, a periodic schedule, which is described in compact form. For large applications, and from a programmer's perspective, this is probably the key advantage.

(iii) 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, which is the common case on non-dedicated Grid platforms for example.

Finally, in addition to the platform model and the scheduling objective described above, our approach enforces the constraint that the divisible load applications are processed fairly and allows for different application priorities.

The rest of this paper is organized as follows. Section 2 details and justifies our platform model. In Section 3, we define the steady-state scheduling problem by deriving the equations that reflect the behavior of the whole architecture/application framework. Section 4 shows that optimizing the throughput is a difficult problem (NP-complete). Section 5 is devoted to the design and analysis of several polynomial heuristics, which are evaluated in simulation in Section 6. Finally, Section 7 concludes the paper and highlights future directions.

2 Platform and Application Model

Our platform model is depicted in Figure 1 and consists of a collection of clusters (or trees of clusters in fact) that are geographically distributed over the internet and interconnected via wide-area links. Each cluster is equipped with a "front-end" processor [15], 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. These links have different bandwidth-sharing properties than local area links, as explained below, and our model allows for a bound on the number of connections that that can be opened by the divisible load applications on these links.

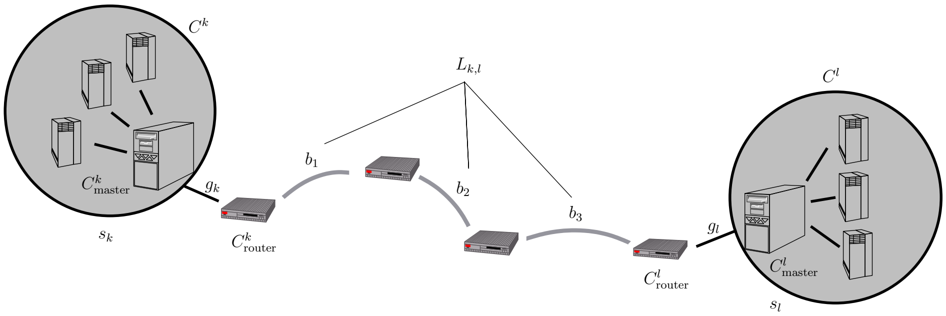

Figure 2: Notations for the platform model.

- 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: $\text{bw}(l_i)$, the bandwidth available for a new connection, and $\text{max-connect}(l_i)$, the maximum number of connections (in both directions) that can be opened on this link. The model for the backbones is as follows. Each connection is granted at most a fixed amount of bandwidth equal to $\text{bw}(l_i)$, 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 [19]. Our model makes it possible to account for this characteristic of wide-area network connections in the context of Grid application scheduling algorithms. 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.

- Our platform consists 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 C^k_{master} and the leaf processors are together "equivalent" to a single processor whose speed s_k can be determined by classical formulas from divisible load theory [30, 6, 4]. In fact, it has also been shown that a tree topology is equivalent to a single processor [6, 5, 7], and thus our model encompasses cases in which the local-area network in each institution is structured as a tree. Consequently, we really 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: Several connections may share the link, but they each receive a portion of the 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.

- 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, which is in particular not assumed to be fully connected. Our model uses realistic bandwidth assignments for each flow: 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 flow.

To the best of our knowledge, the model described above represents the first attempt at modeling relatively complex network topologies along with realistic bandwidth-sharing properties for the purpose of large-scale application scheduling research. We contend that this model, and future evolutions of it, provides a major first step in the development of application-level scheduling strategies that are truly relevant to the new class of platforms brought about by Grid infrastructures.

3 Steady-state scheduling of multiple applications

The steady-state approach has been pioneered by Bertsimas and Gamarnik [12]. Steady-state scheduling allows to relax the scheduling problem in many ways. Indeed, initialization and clean-up phases are neglected, and the emphasis is on the design of a *periodic* schedule. The precise ordering and allocation of tasks and messages are not required, at least in the first step. The main 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 into 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. We outline below the construction step-by-step.

3.1 Steady-state equations

We consider K divisible load applications, A_k, $1 \leq k \leq K$, and cluster C^k initially holds all the input data necessary for application A_k. For each application we define a "payoff factor", π_k, that quantifies the relative worth of the applications. For instance, computing one unit of load per work unit for an application with payoff factor 2 is twice as worthwhile/profitable than for an application with payoff factor 1. This concept makes it possible to implement notions of application priorities for resource sharing. Note that one can set π_k to zero for clusters that do not wish to execute a divisible load application. Similarly, our method is easily extensible to the case in which more than one application originate from the same cluster. We start developing the steady-state scheduling problem with the two following definitions:

- 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 the divisible load application A_l. Reciprocally, portions of the load A_k may be executed by other clusters. We let $\alpha_{k,l}$ be the amount of work 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.
- Cluster C^k opens $\beta_{k,l}$ connections to send the fraction $\alpha_{k,l}$ of the load that is destined to cluster C^l.

The first steady-state equation expresses the fact that a cluster C^k cannot compute more load per time unit than what is allowed by its speed s_k:

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

Indeed, $\alpha_{k,k}$ is the amount of load for application A_k that is processed locally by cluster C^k, and for $l \neq k$, $\alpha_{l,k}$ is the amount of load for application A_l that is exported by cluster C^l to C^k. An interesting feature of steady-state scheduling is that 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 which is 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}:

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

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. Again, there is no need to specify the precise ordering of the communications along the link. Note that we assume that the time to execute a data chunk, or to communicate it along a serial link, is proportional to its size: this amounts to fix the granularity and to manipulate elemental pieces of work. 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 [8].

Next we have to bound the utilization of the backbone links. The third equation deals with the number of opened connections. On each backbone link l_i, there should be no more than max-connect(l_i) different connections used by the divisible load applications:

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

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

$$\alpha_{k,l} \quad \leq \quad \beta_{k,l} \times \min_{l_i \in L_{k,l}} \text{bw}(l_i) \tag{4}$$

The last term in Equation 4 represents the bandwidth allotted to the communication from C^k to C^l: the bandwidth available for a single connection is the minimum of the bw(l_i), taken over all links l_i that constitute the routing 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. One possibility for our objective function is to maximize the total payoff of the computation per time unit, i.e.:

$$\textsc{Maximize} \quad \sum_{k=1}^{K} \alpha_k \times \pi_k. \tag{5}$$

A problem with this objective function is that there is the risk that one application would be unduly favored and granted most of the resources. To achieve a fair balance among all the divisible load applications, one can instead maximize the minimum payoff over all applications, i.e.:

$$\textsc{Maximize} \quad \min_k \{\alpha_k \times \pi_k\}. \tag{6}$$

This maximization corresponds to a MAX-MIN fairness strategy [11] between the different loads, with coefficients π_k, $1 \leq k \leq K$.

The constraints that we have derived earlier and one of the two objective functions defined in Equation 5 or Equation6 form a linear program. For instance, for objective function in

Equation 6:

$$
\begin{array}{l}
\textsc{Maximize} \;\; \min_k \{\alpha_k \times \pi_k\}, \\
\textsc{under the constraints} \\
\left\{
\begin{array}{lll}
\text{(7a)} & \forall C^k, & \displaystyle\sum_l \alpha_{k,l} = \alpha_k \\
\text{(7b)} & \forall C^k, & \displaystyle\sum_l \alpha_{l,k} \leq s_k \\
\text{(7c)} & \forall C^k, & \displaystyle\sum_{l \neq k} \alpha_{k,l} + \sum_{j \neq k} \alpha_{j,k} \leq g_k \\
\text{(7d)} & \forall k, l, & \displaystyle\sum_{l_i \in L_{k,l}} \beta_{k,l} \leq \text{max-connect}(l_i) \\
\text{(7e)} & \forall k, l, & \alpha_{k,l} \leq \beta_{k,l} \times \displaystyle\min_{l_i \in L_{k,l}} \text{bw}(l_i) \\
\text{(7f)} & \forall k, l, & \alpha_{k,l} \geq 0 \\
\text{(7g)} & \forall k, l, & \beta_{k,l} \in \mathbb{N}
\end{array}
\right.
\end{array}
\tag{7}
$$

This program is mixed in the sense that the $\alpha_{l,k}$ are rational numbers and the $\beta_{l,k}$ are integers. Given a platform $\mathcal{P}$ and computational payoffs $(\pi_1, \ldots, \pi_K)$, we define a *valid allocation* for the steady-state mode as a set of values (α, β) such that Equations 7 are satisfied.

Unfortunately, the linear program 7 involves integer variables, the $\beta_{k,l}$, so there is little hope that an optimal solution can be computed in polynomial time. Indeed, we show in Section 4 that determining the optimal throughput of program 7 is a NP-hard problem. However, the linear program 7 captures all the constraints to be satisfied, and we can reconstruct a periodic schedule for every valid allocation, i.e. for every set of values of the $\alpha_{k,l}$ and of the $\beta_{k,l}$ that fulfills the constraints. We derive several heuristics to compute valid allocations in Section 5, and we assess their performances in Section 6.

3.2 Reconstructing a periodic schedule

In this section, we briefly explain how to reconstruct a periodic schedule when given a valid allocation (α, β), i.e. a solution of program 7. Because the divisible load applications are independent, the task is not difficult. We express all the $\alpha_{k,l}$ as $\alpha_{k,l} = \frac{u_{k,l}}{v_{k,l}}$, where the $u_{k,l}$ and the $v_{k,l}$ are integers. The period of the schedule is set to $T_p = \text{lcm}_{k,l}(v_{k,l})$. In steady-state, during each period of length T_p:

- Cluster C^k computes, for each non-zero value of $\alpha_{l,k}$, an integer load $\alpha_{l,k}.T_p$ for 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 $\frac{\sum_l \alpha_{l,k} \cdot T_p}{T_p} \leq s_k$, hence C^k has enough time to process all its load.

- Cluster C^k sends, for each non-zero value of $\alpha_{k,l}$, a data chunk of integer size $\alpha_{k,l} \cdot T_p$ for application A_k, to be processed by cluster C^l during the next period. Similarly, it receives, for each non-zero value of $\alpha_{j,k}$, a data chunk of integer size $\alpha_{j,k}.T_p$ for application A_j, to be processed locally during the next period. All these communications share the serial link, but Equation 2 ensures that $\frac{\sum_{l \neq k} \alpha_{k,l}.T_p + \sum_{j \neq k} \alpha_{j,k}.T_p}{T_p} \leq g_k$, 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 Complexity

In this section we establish a complexity result: optimizing the throughput is NP-hard. We start with the formulation of the associated decision problem, and we proceed to the proof. Note that we cannot use a straightforward reduction from a multicommodity flow problem such as problem ND47 in [2], because there is no prescribed location on where each work should be executed.

Definition 1 (STEADY-STATE-DIVISIBLE-LOAD($\mathcal{P}, \pi, \rho$)). *Given a platform $\mathcal{P}$, computational payoffs $(\pi_1, \ldots, \pi_K)$ and a throughput bound ρ, is there a valid allocation $\mathcal{A}$ such that* $\min_k \{\pi_k \times \alpha_k\} \geq \rho$ *?*

Theorem 1. *STEADY-STATE-DIVISIBLE-LOAD($\mathcal{P}, \pi, \rho$) is NP-complete.*

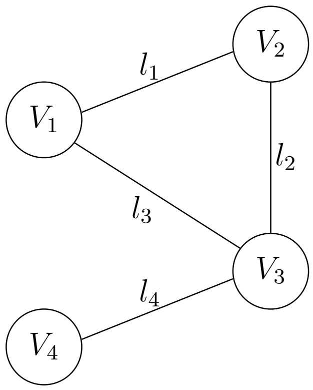

Figure 3: Example of instance $\mathcal{I}_1$ of MAXIMUM-INDEPENDENT-SET

Proof. We first prove that this problem belongs to NP. Given an instance $\mathcal{I}$ of STEADY-STATE-DIVISIBLE-LOAD, we verify that $\mathcal{A} = (\alpha, \beta)$ is a valid solution by checking that Equations 7 are satisfied, and that $\min_k \{\pi_k \times \alpha_k\} \geq \rho$, which can be done in polynomial time.

To prove the completeness of STEADY-STATE-DIVISIBLE-LOAD, we proceed by a reduction from MAXIMUM-INDEPENDENT-SET, which is known to be NP-complete [24]. Consider an arbitrary instance $\mathcal{I}_1$ of MAXIMUM-INDEPENDENT-SET: given a non-oriented graph $G = (V, E)$ and an integer bound B, does there exist a subset V' of V of cardinal at least B and such that no two vertices of V' are joined by an edge of E? From $\mathcal{I}_1$, we construct the following instance $\mathcal{I}_2$ of STEADY-STATE-DIVISIBLE-LOAD:

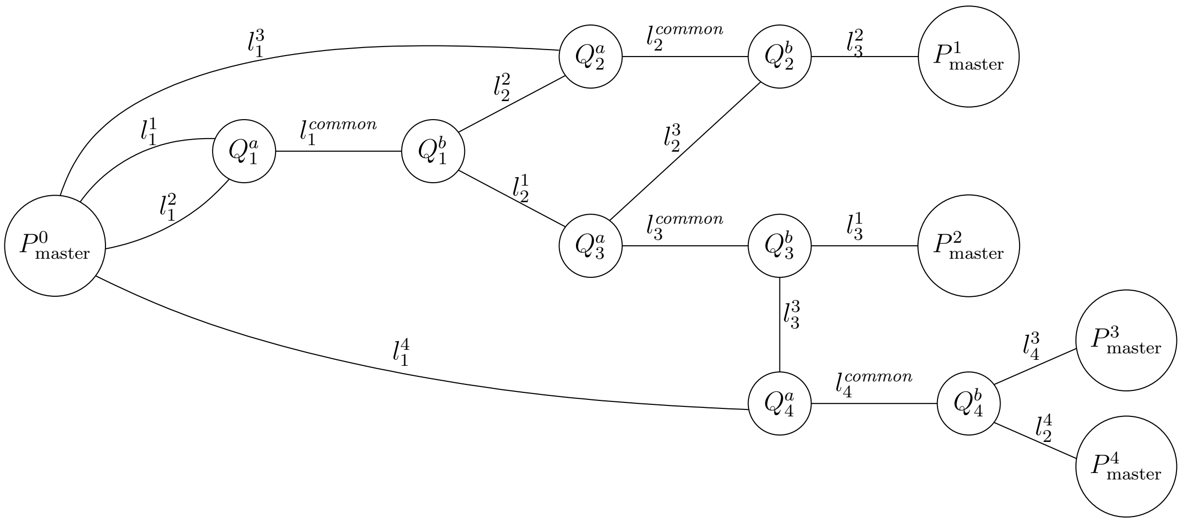

Figure 4: Instance $\mathcal{I}_2$ of STEADY-STATE-DIVISIBLE-LOAD built from the previous instance example $\mathcal{I}_1$

- Let $V = \{V_1, \dots, V_n\}$. The platform of $\mathcal{I}_2$ consists of $n+1$ clusters C^0, C^1, ..., C^n. A set *Route*(i) is associated with each cluster C^i and will be used to determine the routing in the platform as explained below.
- $E = \{e_1, \dots, e_m\}$. For each $e_k = (V_i, V_j) \in E$, we add in the platform graph:
  - two routers Q_k^a and Q_k^b,
  - a backbone link between them: $l_k^{common} = (Q_k^a, Q_k^b)$, with max-connect$(l_k) = 1$ and bw$(l_k) = 1$,
  - a new element k in *Route*(i) and *Route*(j).

  Then, for each set $\mathit{Route}(i) = \{k_1, \dots, k_{|\mathit{Route}(i)|}\}$ associated to cluster C^i, we add the following backbone links, all with max-connect$(l) = 1$ and bw$(l) = 1$: $l_1^i = (C^0, Q_{k_1}^a)$, for $j = 1, \dots, |\mathit{Route}(i)|$, $l_j^i = (Q_{k_j}^b, Q_{k_{j+1}}^a)$ and $l^i_{|\mathit{Route}(i)|+1} = (Q^b_{k_{|\mathit{Route}(i)|}}, C^i)$
- Finally, the routing between cluster C^0 and cluster C^i is given by the following routing path:
$$L_{0,i} = \left\{ l_1^i, l_{k_1}^{common}, l_2^i, l_{k_2}^{common}, \dots, l^{common}_{k_{|\mathit{Route}(i)|}}, l^i_{|\mathit{Route}(i)|+1} \right\} \quad (8)$$
- Cluster C^0 has specific characteristics: $g_0 = n$ and $s_0 = 0$, while all other clusters are such that $g_i = s_i = 1$.

We set the computational payoff to $\pi_0 = 1$ and $\pi_i = 0$ for $j = 1, \dots, n$ (C^0 is the only cluster which has work to do). The throughput bound ρ of $\mathcal{I}_2$ is set to B.

The platform graph that we have constructed has a strong property which is expressed in the following lemma:

Lemma 1. *Two routes (C^0, C^i) and (C^0, C^j) in the platform graph of instance $\mathcal{I}_2$ share a common backbone link if and only if the edge (V_i, V_j) belongs to the graph G of instance $\mathcal{I}_1$.*

Proof. Assume first that edge $e_k = (V_i, V_j)$ belongs to G. Then, by construction, as k is added to the list $Route(i)$ and $Route(j)$, the corresponding link l_k^{common} belongs both to $L_{0,i}$ and $L_{0,j}$: the routes (C^0, C^i) and (C^0, C^j) share the common link l_k^{common}.

Assume now that routes (C^0, C^i) and (C^0, C^j) share a backbone link. According to Equation 8, this link is a l_k^{common} for some k. As $l_k^{common} \in L_{0,i}$ and $l_k^{common} \in L_{0,j}$, then $k \in Route(i)$ and $k \in Route(j)$. The construction of these sets shows that there is an edge e_k between V_i and V_j in G. □

We now prove that there exists a solution to $\mathcal{I}_1$ if and only if there exists a solution $\mathcal{I}_2$:

- Assume that there exists an independent set V' solution of $\mathcal{I}_1$ (so $|V'| \geq B$). From V', we construct the following allocation $\mathcal{A}$:

$$\begin{aligned} \forall i, \quad \alpha_{0,i} = \beta_{0,i} &= \begin{cases} 1 & \text{if } V_i \in V' \\ 0 & \text{otherwise} \end{cases} \\ \forall j \neq 0, \forall i, \quad \alpha_{j,i} = \beta_{j,i} &= 0 \end{aligned}$$

  As V' is an independent set, there is no edge in G between any two vertices of V', so there is no common backbone link between the routes defined by non-zero values of the β's. Each backbone is used by at most one route, and since $\max{-connect} = 1$ for all backbones, Equation 3 is satisfied. There are $|V'|$ (which is less than n) different routes outgoing from C^0, and none incoming to it, so Equation 2 is fulfilled since $g_0 = n$. For all other clusters C^i ($i > 0$), at most one route with bandwidth 1 is incoming and none is outgoing, so Equation 2 is satisfied since $g_i = 1$. Each cluster C^i such that $V_i \in V'$ has to compute an amount of work of 1 unit, which is not more than its speed, so Equation 1 is satisfied.

  Hence, (α, β) defines a valid allocation which reaches the throughput of $|V'| \geq B = \rho$. This is a solution for $\mathcal{I}_2$.

- Assume now that (α, β) is a solution of $\mathcal{I}_2$, which means that it is valid allocation whose throughput is at least ρ. As C^0 has no computing power, it has to delegate the work to other clusters. Each other cluster has a computing speed of one task every time-unit, so there exist at last ρ different routes from cluster C^0 to ρ distinct clusters $C^{k_1}, \dots, C^{k_\rho}$. Since $\max{-connect} = 1$ for each backbone link, only one route can go through each backbone link. Hence, for every couple of routes to clusters C^{k_i} and C^{k_j}, no link is shared, which means that there is no edge (V_i, V_j) in the original graph. So the set of the corresponding vertices $V' = \{V_{k_1}, \dots, V_{k_\rho}\}$ is an independent set in G. As the cardinal of this set is $\rho = B$, V' is a solution of the instance $\mathcal{I}_1$. □

5 Heuristics

In this section we propose several heuristics to solve the multi-application steady state 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.

5.1 Greedy Heuristic

Our greedy heuristic proceeds in a sequence of steps in which resources are allocated to one of the K applications. 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 $\alpha_k \pi_k$ 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 payoff factor π_k.
- Compare the payoff of computing on the local cluster with the payoff of opening a route to the remote clusters. 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.

Let $g_{k,l} = \min\limits_{l_i \in L_{k,l}} \{\mathrm{bw}(l_i)\}$ be the minimum bandwidth available for one connection on a route from cluster C^k to cluster C^l. 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 according to the lexicographic order $\left(\frac{1}{\alpha_k \times \pi_k}, \pi_k\right)$. 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 that can be executed using a single connection:
$$\mathrm{benefit}_m = \min\{g_k, g_{k,m}, g_m, s_m\}.$$
Locally, one can achieve $\mathrm{benefit}_k = s_k$. Select C^l, $1 \leq l \leq K$ so that $\mathrm{benefit}_l$ is maximal. If $\mathrm{benefit}_l = 0$ (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 $\mathrm{alloc} = \mathrm{benefit}_l$ units of load to cluster C^l. If $k = l$ (local computation), allocate only
$$\mathrm{alloc} = \max_{m \neq k} \{\min\{g_k, g_{k,m}, g_m, s_k\}\}$$
units of load. This last 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: $s_l \leftarrow s_l - \mathrm{alloc}$
   - Allocate work: $\alpha_{k,l} \leftarrow \alpha_{k,l} + \mathrm{alloc}$

- In case of a remote computation (if $k \neq l$) update network characteristics:

$$\begin{aligned}
&\forall l_i \in L_{k,l},\ \text{max-connect}(l_i) \leftarrow \text{max-connect}(l_i) - 1\\
&g_k \leftarrow g_k - \text{alloc}\\
&g_l \leftarrow g_l - \text{alloc}\\
&\beta_{k,l} \leftarrow \beta_{k,l} + 1
\end{aligned}$$

7. Go to step 2.

5.2 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 STEADY-STATE-DIVISIBLE-LOAD problem, while an LP formulation allows rational $\beta_{k,l}$ and gives an *upper bound* on the 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 this 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.

5.2.1 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:

$$\begin{aligned}
\forall k,l, \qquad & \widehat{\beta}_{k,l} = \lfloor \widetilde{\beta}_{k,l} \rfloor,\\
\forall k,l, \qquad & \widehat{\alpha}_{k,l} = \min \left\{ \widetilde{\alpha}_{k,l}, \lfloor \widetilde{\beta}_{k,l} \rfloor \min_{l_i \in L_{k,l}} \text{bw}(l_i). \right\}
\end{aligned}$$

With these new values, we have:

$$\begin{aligned}
\forall k,l, \qquad & \widehat{\beta}_{k,l} \leq \widetilde{\beta}_{k,l},\\
\forall k,l, \qquad & \widehat{\alpha}_{k,l} \leq \widetilde{\alpha}_{k,l},\\
\forall k,l, \qquad & \widehat{\alpha}_{k,l} \leq \min_{l_i \in L_{k,l}} \text{bw}(l_i),
\end{aligned}$$

which shows that we have obtained a solution to the linear program in which all $\beta_{k,l}$ take integer values. We label this method LPR.

5.2.2 LPRG: Round-off + Greedy

Obviously, rounding down all the $\beta_{k,l}$ variables with LPR may lead to a very poor result as the remaining network capacity is lost. The LPRG heuristic reclaims this residual capacity by applying the technique described in Section 5.1. Intuitively, LPR gives the basic framework of solution, while the Greedy heuristic refines it.

5.2.3 LPRR: Randomized Round-off

Relaxing an integer linear program into rational numbers is a classical approach, and several solution have been proposed for rounding. Among others is the use of randomized approximation. In [26, 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 a good probability to a feasible solution that achieves the optimal throughput. Although this theoretical result seems attractive, it has some drawbacks. First, as mentioned in Section 4, 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 [26, 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 no big deal), or it provides a solution which does not satisfy all the constraints, which forces us to modify the solution to make it feasible.

Coudert and Rivano proposed in [21] a rounding heuristic based on the method of [26, chapter 11] in the context of optical networks. Their method seems more practical as it always provides a feasible solution. We use the same approach as in [21] to build an algorithm based upon randomized rounding. This heuristic, the LPRR heuristic, 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 following constraint to the linear program: $\beta_{k,l} = v$.
5. If there exists at least a route k, l for which no $\beta_{k,l}$ value has been assigned yet, go to step 2.

We point out that LPRR solves K^2 linear programs, and is thus much more computationally expensive that the other LP-based heuristics. We present results comparing the effective complexity of all our heuristics in Section 6.

6 Experimental results

In this section, we use simulation to evaluate the G, LPR, LPRG, and LPRR heuristics described in Section 5 when applied to the STEADY-STATE-DIVISIBLE-LOAD problem. Solving the mixed LP problem for the optimal solution takes exponential time; consequently we cannot use it in practice and cannot compare our heuristics to the optimal. Instead we use the solution to the rational LP problem as a comparator, as it provides an upper bound on the optimal solution (it cannot be achieved/used in practice as $\beta_{k,l}$ values need to be integer). We denote this method by LP. We solve the scheduling problem for the two objective functions given in Equations 5 and 6, which we denote by SUM and MAXMIN respectively.

parameter name	values
K	$5, 15, \ldots, 95$
connectivity	$0.1, 0.2 \ldots, 0.8$
heterogeneity	$0.2, 0.4, 0.6, 0.8$
mean g	$50, 250, 350, 450$
mean bw	$10, 20, \ldots, 90$
mean $maxcon$	$5, 15, \ldots, 95$

Table 1: Parameter settings used for simulation experiments.

We instantiate random platform models based on the following parameters: K, the number of clusters in the network; *connectivity*, the probability that any two clusters are connected; g, the local bandwidth on the local link in a given cluster; bw, the bandwidth of one connection on a given backbone link; $maxcon$, and the maximum number of connections on a given backbone link. The last three parameters are chosen according to a *heterogeneity* parameter that denote the difference (i.e., the relative ratio) between platform components in the same platform. More specifically, parameters g, bw, and $maxcon$ were sampled from a uniform distribution between $mean * (1 - heterogeneity)$ and $mean * (1 + heterogeneity)$, where the *mean* values are given in Table 1. This table also gives the range of values used for K, *connectivity*, and *heterogeneity*. Since only relative values are meaningful in a periodic schedule, we fix the computing speed at 100. We generated 10 random platforms for each setting of these parameters, totaling 269,835 different platform configurations.

We ran LP, G, LPR, LPRG on each of these topologies. As LPRR takes a long time to execute (it solves K^2 linear programs, see Section 6.3), we only evaluate it on 80 of our topologies.

6.1 Comparison of G, LPR, and LPRG

Our simulation results clearly show that LPR exhibits very poor performance when compared to both G and LPRG. Typically LPR does not utilize a significant portion of the network capacity, and in some cases all $\beta_{k,l}$ values are rounded down to 0, leading to an objective value of 0.

More interesting is the comparison between G and LPRG. Over all the platforms that we evaluated, the ratio of the objective values achieved by LPRG to that by G is: 1.98 for MAXMIN and 1.02 for SUM. For a closer look, Figure 5 plots the average ratio of the objective values achieved by LPRG and G to that achieved by LP, which is an upper bound on the optimal, versus the number of clusters K.

We see that LPRG always achieves higher SUM values than G, and as K gets larger, the advantage grows larger. At large number of clusters ($K \approx 80$), LPRG is very close to the upper bound on the optimal solution, which leads us to believe that LPRG is likely very close to the optimal. At $K = 5$, LPRG obtains a MAXMIN value slightly lower than G, but its MAXMIN value becomes increasingly higher than that of G as K grows. Both LPRG and G achieve a high SUM value when K is large, but their MAXMIN value is much lower, $\approx 65\%$ of the LP upper bound at $K = 75$. We will see in Section 6.2 that the LPRR heuristic does much better in this area.

We have mined our results to identify potential trends about how platform characteristics impact the relative performance of our heuristics. No clear trend emerges in the MAXMIN

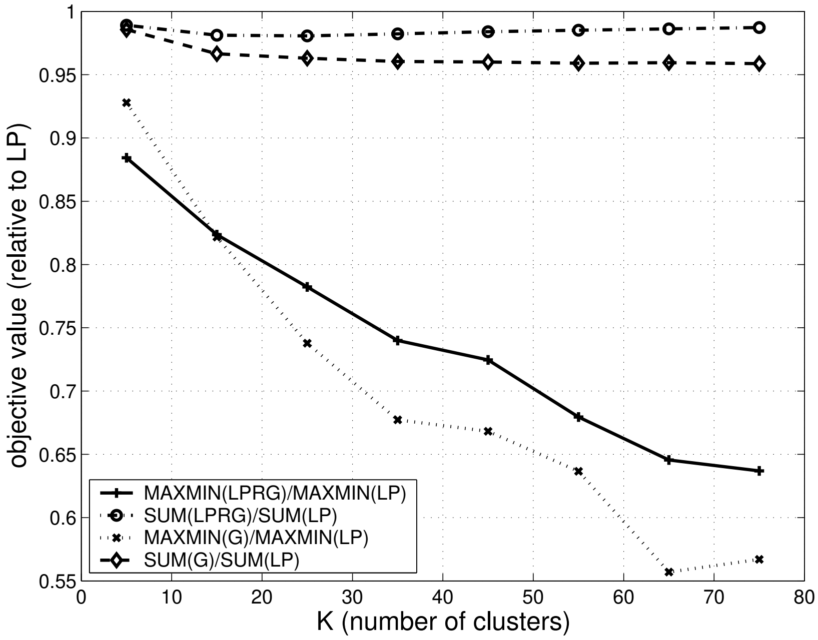

Figure 5: Comparison of LPRG and G, with different K

case and the relative performance of G and LPRG is rather irregular, with LPRG leading to better results in the vast majority of the cases however. The relative performance of G and LPRG is more regular in the SUM case, but we found that variations in platform parameters besides K (i.e., *connectivity*, *heterogeneity*, g, bw, or $maxcon$) does not lead to significant variations in relative performance.

6.2 LPRR vs LPRG

As seen in the previous section, LPRG achieves low MAXMIN value at large K ($\sim 35\%$ lower than the LP upper bound). The LPRR heuristic achieves better results, at the cost of a higher complexity (K^2 times larger than LPRG). For the small subset of 80 topologies we tested, Figure 6 shows that LPRR achieves objective values very close to the upper bound. While LPRR rounds off the $\beta_{k,l}$ values to the closest integer with higher probability, we also tested another version that rounds off up or down randomly with equal probability. It is interesting to note that this version performed much worse than LPRR.

6.3 Running Time of Heuristics

It is known theoretically that the Simplex method for solving LP takes $\binom{n+m}{m}$ time in the worst case, where m is the number of basic variables, and n non-basic variables. In practice, this method is one of the fastest ones for solving linear programs. We used the `lp_solve` package [9], which implements the Simplex algorithm. Since the actual running time of this LP solver is drastically different than the theoretical worst-case bound, we can only compare the running time of our LP-based heuristics empirically. We measured the running time of LP, G, LPR, LPRG, and LPRR on a set of 112 topologies with $K = 10, 20, 30, 40$, on a Pentium III 800MHz computer. Figure 7 plots heuristic running times vs. K, on a logarithmic scale.

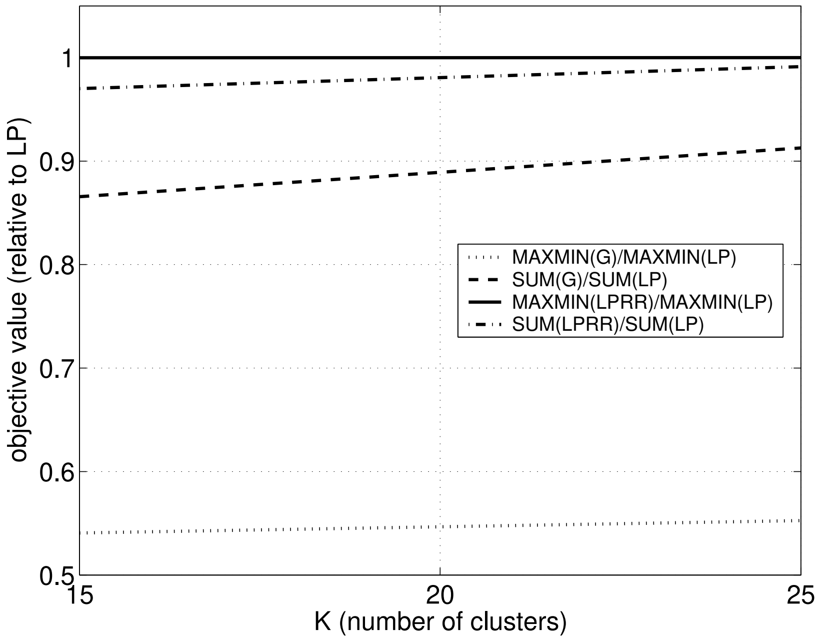

Figure 6: Comparison of LPRR and LPRG, 80 topologies

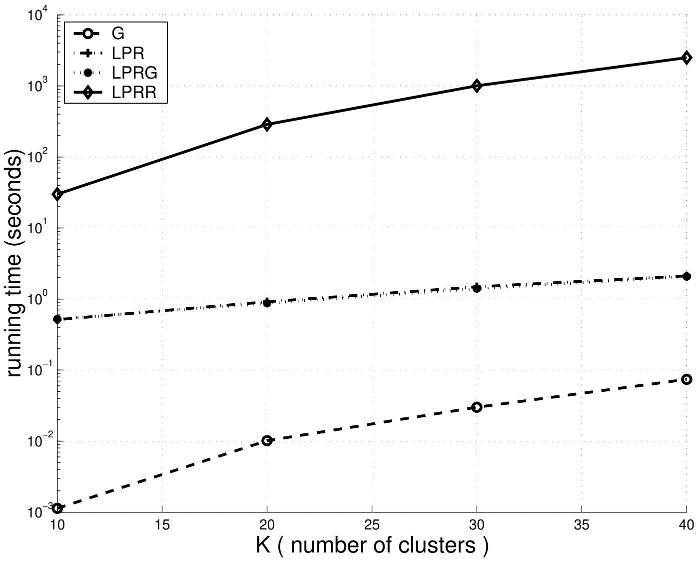

Figure 7: Running time of G, LPR, LPRG, and LPRR, **log scale**.

We can see that, expectedly, G runs significantly faster than the LP-based heuristics, taking at most 0.1 seconds. In essence, G works like the Simplex method but uses a more selective scheme for "tightening" the constraints. LP, LPR and LPRG are slower, and their running time increases *linearly* from 0.5 seconds for $K = 10$ to 2 seconds for $K = 40$. By contrast, LPRR's running time increases by K^2, and is approximately 1000 times larger than our other heuristics at $K = 40$. Although we focus on steady state performance and thus an infinite execution, LPRR is most likely impractical as in a real-world scenario the scheduling process must typically be executed periodically to adapt to changing platform conditions and/or application loads.

7 Conclusion

In this paper we have addressed the steady-state scheduling problem for multiple concurrent divisible applications running on Grid 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 (in fact some Grids run primarily divisible load applications [18]). However, only a few authors had explored the simultaneous scheduling of multiple such applications on a distributed computing platform [14, 34] and in this paper we have made the following contributions. First, we used a realistic platform model that captures some of the fundamental network properties of Grid platforms. Then, we formulated our scheduling problem as a mixed integer-rational linear program and proposed two objective functions, SUM and MAXMIN. Both implement a notion of weighted priorities for resource sharing between applications, and MAXMIN ensures some notion of fairness. We proposed a greedy heuristic, G, and three heuristics based on the solution to the linear program: LPR, which simply rounds down the rational solution to the linear program; LPRG, which uses the G heuristics to refine the solution to the linear program; and LPRR, which uses randomized rounding. We evaluated these heuristics with extensive simulation experiments for large number of random Grid configurations. We compared our heuristics to an upper-bound on the optimal schedule which is obtained by solving the purely rational version of our mixed linear program. We found that although LPR leads to very poor performance, LPRG achieves performance close to the optimal for the SUM objective function and in fact performs better than G. For the MAXMIN objective function, LPRG only achieves 65% of the upper-bound on the optimal performance for a large number of clusters. In this case, LPRR achieves performance very close to the optimal, but at the cost of a much higher complexity than LPRG (approximately a factor K^2, where K is the number of clusters in the platform).

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 [13, 32]. We have gathered such information as part of other research projects. 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 [27, 19]) provides the foundation for refining our network model, both based on empirical measurements and on theoretical modeling of network traffic. Finally, one could envision extending our application model to address the situation in which each divisible load application consists of a set of tasks linked by dependencies. This would be an attractive extension of the mixed task and data parallelism approach [20, 28, 3, 33, 17] to heterogeneous clusters and grids.

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] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer, Berlin, Germany, 1999.

[3] H. Bal and M. Haines. Approaches for integrating task and data parallelism. *IEEE Concurrency*, 6(3):74–84, 1998.

[4] 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.

[5] G.D. 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.

[6] S. Bataineh, T.Y. 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, October 1994.

[7] 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, September 2003.

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

[9] Michel Berkelaar. LP_SOLVE. http://www.cs.sunysb.edu/~algorith/implement/lpsolve/implement.s

[10] F. Berman, G. Fox, and T. Hey, editors. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley PUblishers, Inc., 2003.

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

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

[13] Michael D. Beynon, Tahsin Kurc, Alan Sussman, and Joel Saltz. Optimizing execution of component-based applications using group instances. *Future Generation Computer Systems*, 18(4):435–448, 2002.

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

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

[16] 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.

[17] T. D. Braun, H. J. Siegel, and N. Beck. Optimal use of mixed task and data parallelism for pipelined computations. *J. Parallel and Distributed Computing*, 61:810–837, 2001.

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

[19] 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.

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

[21] 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.

[22] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, Inc., San Francisco, USA, 1999.

[23] I. Foster and C. Kesselman, editors. *Computational Grids: Blueprint for a New Computing Infrastructure*. M. Kaufman Publishers, Inc., 2nd edition, 2003.

[24] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1991.

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

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

[27] 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.

[28] S. Ramaswamy, S. Sapatnekar, and P. Banerjee. A framework for exploiting task and data parallelism on distributed memory multicomputers. *IEEE Trans. Parallel and Distributed Systems*, 8(11):1098–1116, 1997.

[29] T.G. Robertazzi. Divisible Load Scheduling. URL: http://www.ece.sunysb.edu/~tom/dlt.html.

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

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

[32] 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.

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

[34] H.M. Wong, D. Yu, , V. Bharadwaj, and T.G. 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.

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