2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing

Planning Large Data Transfers in Institutional Grids

Fatiha Bouabache
Universite Paris Sud-XI, F-91405; INRIA
Email: fatiha.bouabache@lri.fr

Thomas Herault
Universite Paris Sud-XI, F-91405; INRIA; University of Tennessee
Email: thomas.herault@lri.fr

Sylvain Peyronnet
Universite Paris Sud-XI, F-91405; INRIA
Email: syp@lri.fr

Franck Cappello
INRIA; University of Urbana Champaign; Universite Paris Sud-XI, F-91405
Email: franck.cappello@lri.fr

***Abstract*—In grid computing, many scientific and engineering applications require access to large amounts of distributed data. The size and number of these data collections has been growing rapidly in recent years. The costs of data transmission take a significant part of the global execution time. When communication streams flow concurrently on shared links, transport control protocols have issues allocating fair bandwidth to all the streams, and the network becomes sub-optimally used. One way to deal with this situation is to schedule the communications in a way that will induce an optimal use of the network.**

We focus on the case of large data transfers that can be completely described at the initialization time. In this case, a plan of data migration can be computed at initialization time, and then executed. However, this computation phase must take a small time when compared to the actual execution of the plan. We propose a best effort solution, to compute approximately, based on the uniform random sampling of possible schedules, a communication plan. We show the effectiveness of this approach both theoretically and by simulations.

***Keywords*-Scheduling, Data Transfer**

I. Introduction

A Grid is an infrastructure consisting of the aggregation of several distributed resources, usually from different administrative domains. We focus on cluster of clusters, i.e. grids built by companies and universities by aggregating the resources of several clusters. Users expect to obtain, from the use of such a grid, larger systems able to address the complexity of their problems. One of the features of a Grid is its size, orders of magnitude larger than a single cluster.

In grid computing, many applications require access to large amounts of distributed data. More specifically, a major source of large data transfer comes from a characteristic of large scale grids. Having a high number of components also implies that the overall probability of failure of one of the multiple components of the Grid is high. Hence, fault tolerance is a feature of Grids recognized as critical. Rollback recovery is the most common fault tolerance technique used in High Performance Computing and scientific applications. This technique relies on the reliability of the checkpoint storage system to ensure fault tolerance. To ensure checkpoint storage reliability of coordinated rollback/recovery protocols, we proposed to replicate the checkpoint images over a set of checkpoint servers distributed over the different clusters [1]. Storing these images in the same cluster they were generated is not reliable, because the link between two clusters may be broken (due to a failure of a gateway for example) or overloaded. During a checkpoint wave and especially during the replication step, huge amounts of data flow between the different components, and between clusters, which may increase the checkpoint wave completion time, and introduce unusual loads on the links between clusters.

A key point of an efficient data transfer is to ensure that the network protocol used to control the transfers will not enter a "thrashing" state, where the allowed bandwidth is drastically reduced because the system is unable to evaluate the requirements of the application. It has been proven that scheduling of (TCP) communications in Grids, or wide-area networks in general, is fundamental for efficiency reasons.

To do this, different solutions exist (see section II). However, most of these solutions are interactive and adapt their execution during the transfer. In our case, the location of the data and the destination goal are both given by the replication strategy, and data are available almost simultaneously due to the coordination of the checkpoint mechanism. Hence, a global communication plan can be computed, that takes as input the two configurations, and the system needs to compute a data migration plan that convert the initial configuration (before replication) to the final configuration (after replication). This migration schedule tries to minimize the time taken to complete the whole migration.

Such planning problem is widely believed to be computationally intractable. Exhaustive approaches are subject to exponential combinatorial explosion. A classical approach to deal with such phenomenon is to design highly efficient but approximate algorithms. In this paper, we present an approximation algorithm based on the uniform random sampling of possible schedules. We show the effectiveness of this approach both theoretically and by simulation.

978-0-7695-4039-9/10 $26.00 © 2010 IEEE
DOI 10.1109/CCGRID.2010.68

II. Related work

The problem of scheduling concurrent huge data transfers in Grid technology has been addressed from different point of views in the literature.

Network bandwidth is one of the primary parameters that impact the performance. The paper [2] considers the problem of huge data transfers and bandwidth sharing in grid infrastructure, where transfer delay bounds are required. This study is based on a scheduling service controlling the starting time and the congestion level. Marchal et al. in [3] addresse the problem of optimal resource sharing to avoid potential bottlenecks. Two resource request scenarios have been identified, aiming at optimizing the request accept rate and resource utilization. The optimization problems, proven NP-complete, are then solved by heuristic algorithms.

The resource reservation technique in general and bandwidth reservation specially is widely used to manage the bulk data transfer, whether to avoid congestion or to minimize the transfer time. In general there are two types of reservation: immediate reservations made just in time and advance reservations which allow to reserve a resource a long time before its use. However, the most useful technique in grid computing is the advance reservation. Several solutions consider the data transfer problem using the QoS and the resource reservation issues. Zhang et al. [4] propose an agreement-based data transfer service, providing some QoS guarantees. The service allows the client to express agreements for quality of data transfer. Those agreements are based on certain QoS metrics, such as the transfer time. Zhang et al. evaluate three prototype implementations of this service using mechanisms of traffic prediction, rate-limiting and priority based adaptation.

Flow scheduling is one of new alternative solutions to traditional QoS and reservation. In [5], Marchal et al. propose to schedule the data transfer requests by tuning the transmission window, such that the request accept rate and network resource utilisation will be optimized. They propose a bandwidth sharing optimisation objective specific to the grid context. The optimisation problem was proven to be NP-complete, so heuristics was proposed and studied using simulations. Authors of [6] study the bulk data transfer scheduling problem. The goal is to find the optimal bandwidth allocation profile for each task minimizing the overall congestion. They propose a multi-interval scheduling, where the active window of a task is divided into multiple intervals, and bandwidth value is assigned independently in each of them. They prove that this technique is sufficient and necessary to attain the optimum. Specifically, they show that the problem can be solved as Maximum Concurrent Flow Problem in a polynomial time. In the same direction, [7] addresses the need of network dimensioning for bulk data transfers in Grid networks and introduces an $M/M/1/M - RPS$ queue model to predict the blocking probability of bulk data transfer requests. The different flow scheduling solutions are generally based on runtime end-hosts mechanisms, controlling flow sending time and rate. [8] explore the end-host based mechanisms. It quantifies and compares the end-host based mechanisms and studies their interaction with transport protocols.

Recently, another approach proposed in [9] has described somes links between maximum-flow computations and sub-modular energy minization. Such an approach could give some insights in solving the same challenge via another approach.

Compared with these scheduling solutions, we consider a public internet network, where source routing is not necessarily enabled, so it is a complex task to control all the path connecting two nodes. Moreover, we consider a checkpoint image storage so that all the different request are present at the same time and computing a plan is possible. There is a global goal, which is to reduce the global time taken to terminate all the transfers.

III. Scheduling Data Transfers

Our grid spans multiple domains and uses Internet to connect between the different clusters. On such environment there is a set of system constraints that we cannot control, eg. we have no control over the routing tables and the failures that can occur. So it is a complex task to control all the path connecting two nodes. For this reason, we propose a best-effort solution: we control only which source node is allowed to send to its destination node (following a communication path depending on the routing tables of the system) at a given time. We assume that the routing tables of the system balance the load of the communication between the different links, and that bandwidth sharing is possible.

We model our problem as a multiflow network: Let $G = (V, E)$ be a flow network represented by a directed graph, where V is the set of nodes in the network and E the set of the different edges. Each edge $(u, v) \in E$ has a capacity $c(u, v) > 0$. $c(u, v)$ is the maximum amount of data that can pass through the edge connecting u to v. $\mathcal{S} = \{0, 1, ..., n - 1\}$ and $\mathcal{T} = \{t_0, t_1, ..., t_n - 1\}$ are two disjoint subsets of V, representing, respectively, the sources and sinks in the Network. They have the same cardinal n representing the number of flows that have to be transferred. f_i is the flow that has to be sent from s_i to t_i. f^i_{uv} is the flow i pushed from u to v, thus we have the capacity constraint $\sum_{i=0}^{n-1} f^i_{uv} \leq c_{uv}$ for each $(u, v) \in E$.

We denote by R the set of data transfer requests, and we consider $t(s)$ the function that, for each source s, gives the corresponding sink. We consider a set of data transfer requests represented by pairs (s, ν_s), where s is the source and ν_s the volume of data that has to be transferred from the source s to the destination $t(s)$. The number of data transfer requests is equal to the number of sources

$card(R) = card(\mathcal{S})$, and each request represents a different data transfer task.

$\mathcal{P}(R)$ denotes the set of all permutations of all the elements of the set R. The cardinal of $\mathcal{P}(R)$ is $n!$. Each permutation vector τ_i of $\mathcal{P}(R)$ contains all the elements of R. A permutation defines an order of priority on the sources: if (s_i, ν_{s_i}) appears before (s_j, ν_{s_j}) in a permutation τ, then the source s_i will be considered for transmission before the source s_j in this permutation.

Our goal is to find an optimal order of the sources, for moving all data from all sources to all destinations, that minimize the time taken to complete all the migrations. So, we want with our scheduling multiflow algorithm to find the optimal permutation that minimizes the time taken to complete the migration or the execution of tasks in R.

The exhaustive scheduling multiflow algorithm (3) is based on two main functions the $flow$ function (1) and the $TimeMultiflow$ function (2).

Algorithm 1 $flow(G, s)$

Require: G: the flow network.
Require: s: a source
Computes: (RG, c), where RG is the residual graph and c is the capacity of the maximal flow with source s on flow network G
Use the push-relabel algorithm [10]

The role of the flow function is to compute, for each different combination, the quantity of data that can be transferred on one step by each source within the combination. We base our $flow$ function on the classical $push - and - relabel$ maxflow algorithm [10]. While the general algorithm has $O(V^2E)$ time complexity, the implementation with FIFO vertex selection rule has $O(V^3)$ running time. This algorithm takes as an input the flow graph $\mathcal{G}$ and the source s, and computes the quantity of data that can be transferred from s to $t(s)$ on one step. In our case, the $flow$ function gives also the residual graph.

The goal of the $TimeMultiflow$ function is to compute for each combination the time taken to execute all the different tasks according to their order in this combination. So, it takes as an input the graph G and the combination that it has to study τ_i. This function gets the different couples within τ_t, one by one, from $\tau_t[0]$ (the greatest priority) to $\tau_t[n-1]$ (the lowest priority). For each couple, it calls the $flow$ function that take the corresponding source s and the graph G. As we can see, only the first call will be executed on the initial graph, the following calls are executed on the residual graph RG. When the capacity $C(s)$ of this source s is computed, the $TimeMultiflow$ function computes the time $T(s)$ needed to transfer ν_s. Once all the couples within τ_t are considered, the time these concurrent flows can be sustained is computed, as the smallest time it takes to exhaust one of the sources. Then, only the couples that have a transmission time greater than the smaller one are put in the new τ_t according to the initial order in τ_i. For each couple in the new τ_t, the data volume that has to be transferred is updated by subtracting the amount of data that was transferred during the previous step. This is executed until τ_t becomes empty. τ_t eventually becomes empty, because if any couple (s, ν_s) remains in τ_t, at some iteration, s is the only source with a non-negative ν_s, thus in the next iteration, this source must be selected and can be selected, because the algorithm restarts from the initial flow graph. At the end, the $TimeMultiflow$ function returns a pair of the total transmission time and the corresponding initial combination (T_{total}, τ_i).

Algorithm 2 $TimeMultiflow(G, \tau_i)$

Require: G: the flow network.
Require: τ_i: sequence of couples (s, ν_s), s being a source, ν_s the size of the data to transmit from this source.
Computes: T_{total}, the time used to transfer all data from all sources to all sinks, following the order of τ_i.

```
T_total ← 0
τ_t ← τ_i
while τ_t ≠ ∅ do
    RG =← G
    for all (s, ν_s) ∈ τ_t do
        (RG, c) ← flow(RG, s)
        C(s) ← c
        T(s) ← ν_s / C(s)
    end for
    T ← min_{s∈τ_i}{T(s)}
    T_total ← T_total + T
    τ_t ← {(s, ν_s − T.C(s)) / (s, ν_s) ∈ τ_t ∧ ν_s − T.C(s) ≠ 0}
end while
return (T_total, τ_i)
```

The goal of the scheduling multiflow algorithm is to find the smallest transmission time and the associated combination. It takes as inputs G and R. It initializes the mintime T_{min}. Then, for each combination $\tau_i \in \mathcal{P}(R)$, it calls the $TimeMultiflow$ function that computes the time needed to execute all the data transfer requests. It then updates the minimum time T_{min}. When all the combinations have been considered, it returns the minimum time and the corresponding combination (T_{min}, τ_{min}).

IV. Approximate Scheduling Algorithm

As said above, the exhaustive algorithm to find the combination that leads to the best transfer time is subject to combinatorial explosion. This means that regardless the time complexity of the algorithm, the size of the input is so large that the problem becomes practically intractable. A classical approach to deal with such phenomenon is to design highly efficient but approximate algorithms. Highly efficient means that either the complexity is sub-linear (most

Algorithm 3 $ExhautiveMultiflow(G, R)$

Require: G: the flow network
Require: R: a set of couples (s, ν_s), where s is a source, and ν_s the size of the data this source wants to transmits
Computes: the minimal time needed to transmit this data

```
T_min ← +∞
for all τ_i ∈ P(X) do
  (T, τ) ← TimeMultiflow(G, τ_i)
  if T < T_min then
    T_min ← T
    τ_min ← τ
  end if
end for
return (T_min, τ_min)
```

of the time polynomial in the logarithm of the input size) or even independent of the size of the input.

In this section, we propose an approximation algorithm based on the uniform random sampling of possible schedules. The algorithm is described formally in Figure 4. The principle is quite simple. It consists in repeating a given number of time the following process: a combination of the sources is chosen uniformly at random. Then the transfer time for this combination is computed. If this transfer time is lower than previously computed ones, then it becomes the potential lower transfer time and the associated ordering of sources the best combination. After a while, the minimum from the sampled transfer times (and associated combination) is considered the best choice. The critical point of the algorithm is thus the computation of the number of randomly chosen combinations that are needed to obtain one of the smallest value. We call this number the sample size of the approximation algorithm.

This size is independent of the size of the input (the size of the graph). However, it depends on two new parameters. The *accuracy* parameter ε defines the quality of the result of the approximation algorithm, while the *confidence* parameter, δ, gives the confidence in the result.

Intuitively, these two parameters mean that the output of the algorithm falls into the fraction ε of smallest values with probability $1 - \delta$. δ is typically set to a very small value, meaning that the probability of our algorithm to give a correct answer is close to 1. The setting for ε is highly dependent of the trade-off between the average transfer time and the time that can be spent in finding a better one. The sample size used in our algorithm is $\frac{1}{\varepsilon} \cdot \ln(\frac{1}{\delta})$.

The following lemma proves the correctness of our algorithm with this sample size. The density of a given class C corresponds to the fraction of values we consider into a set of n values. For us it means that we are sure that the output of the approximation algorithm is in the $\varepsilon \cdot |n|$ smallest transfer times (where n is the number of different possible transfer times).

Algorithm 4 $ApproximateMinTime(G, R, \delta, \varepsilon)$

Require: G: the flow network
Require: R: a set of couples (s, ν_s), where s is a source, and ν_s the size of the data this source needs to transmits.
Require: δ: confidence parameter
Require: ε: accuracy parameter
Computes: an approximation of the minimal time needed to transmit this data with accuracy ε, and confidence δ

```
T_min ← +∞
N ← (1/ε) · ln(1/δ)
for i ← 1..N do
  pick τ ∈ P(R) uniformly at random
  (T, τ) ← TimeMultiflow(G, τ)
  if T < T_min then
    T_min ← T
    τ_min ← τ
  end if
end for
return (T_min, τ_min)
```

Lemma 1: Suppose that the density of a given class C of items is ε (that is there are $\varepsilon \cdot n$ elements of this class in the whole set of n items). Then, by drawing uniformly at random $\frac{1}{\varepsilon} \cdot \ln(\frac{1}{\delta})$, the probability of having at least one element of the class C is $1 - \delta$.

Proof: We compute the probability not to found such an item. Let denote by E the event of not drawing an item from C after drawing s random items. We are interested in the value of s.

$$prob[E] = (1 - \varepsilon)^s \leq e^{s \cdot \varepsilon}$$

if we take $s = \frac{1}{\varepsilon} \cdot \ln(\frac{1}{\delta})$, then we obtain:

$$prob[E] \leq e^{-(1/\varepsilon) \cdot \varepsilon \cdot \ln(\frac{1}{\delta}} \leq \delta$$

Since the event we are interested in is the complementary event to E, then the result holds.

■

V. Evaluation

On the one hand, the goal of this section is to show that with the sampling approach of the approximate algorithm, we can find a combination that gives a transfer time close to the optimal. To prove that we are close to the optimal, we execute the exhaustive scheduling algorithm so that we can have all the different possible transfer time. On the other hand, one of the main objectives of this study is to examine our protocol behavior in realistic architecture which approaches the structure of the Grid as closely as possible. However, with the exhaustive scheduling algorithm, the number of possible schedules explodes. To overcome this problem, we use at first a synthetic graph to study the effectiveness of the approximate algorithm.

A. Synthetic graph

We consider the synthetic graph depicted in figure 1. We note this graph $\mathcal{SG}_{11} = (V, E)$ where $card(V) = 42$ and $card(E) = 50$. In this graph, sources belong to a set $\mathcal{S} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$ and destinations to $\mathcal{T} = \{t(0), t(1), t(2), t(3), t(4), t(5), t(6), t(7), t(8), t(9), t(10)\}$.

Each source s has to transfer $\nu_s = 20000Mb$, so we have 11 different transfer tasks: $card(R) = 11$. There is $39,916,800$ different task combinations ($11! = 39,916,800$).

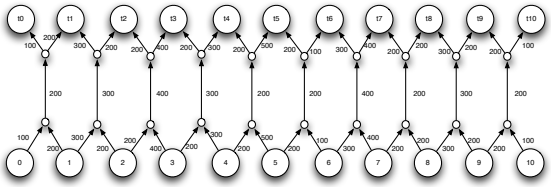

Figure 1. Synthetic graph $\mathcal{SG}_{11}$

The exhaustive algorithm

Figure 2 shows the distribution of transfer times in the task combinations space of $\mathcal{SG}_{11}$. For each possible combination τ_i of tasks, we compute the result of $TimeMultiflow(\mathcal{SG}_{11}, \tau_i)$, and count the number of combinations producing a given time. We plot the number of task combinations on a logarithmic scale, and we observe that the transfer time ranges from 62s to 490s (factor of roughly 8 between the smallest and the highest). The goal of our algorithm is to find the smallest transmission time by sampling randomly source combinations. Since some transfer times have a very high probability to appear, a single random choice is likely to give it. In this case, that would give a transfer time between 200s and 300s. Still, it is noticeable that transfer times lower than 100s have a cumulated number of occurrences high enough to be chosen.

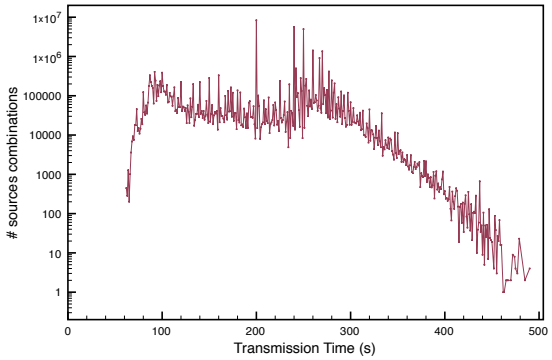

Figure 2. Number of sources combinations vs. transmission time, for $\mathcal{SG}_{11}$

The approximate algorithm

Figure 3 represents the probability to find a combination of tasks with a transfer time lower than a given time, as function of the size of samples (N in Section IV) used to iterate in the approximate algorithm. This probability has been estimated as the mean value of 1,000 runs of the algorithm.

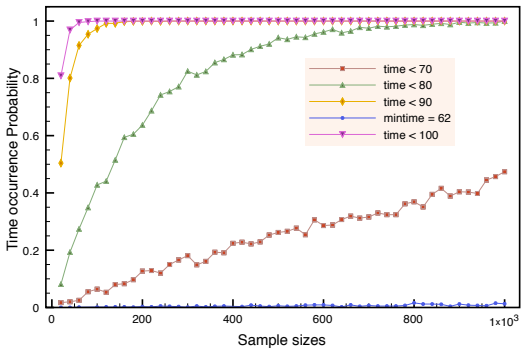

Figure 3. Probability to find a combination of sources with a transmission time lower than a given time, as function of the size of samples for $\mathcal{SG}_{11}$

This figure illustrates that even with a sample including a low number of combinations, the algorithm provides with high probability a transfer time much lower than the average transfer time. It also shows a trade-off between the sample size and the performance of the scheduling. A larger sample implies a lower transfer time, but also a larger overhead to compute the schedule. One can see that when we try to get a better approximation, closer to the optimal time of 62s, the probability to find a combination that corresponds to this time increases very slowly with the sample size.

This is illustrated by Figure 4. This figure is the same representation as Figure 3, with a single goal: the optimal schedule ($mintime = 62s$). One can see that a very large sample is mandatory to find this schedule with a high probability. This is inherent to sampling methods: it is very costly to find the optimal value, but very cheap to find values close to the optimal. This is confirmed by the theoretical result obtained from the formula $\frac{1}{\varepsilon} \cdot \ln(\frac{1}{\delta})$.

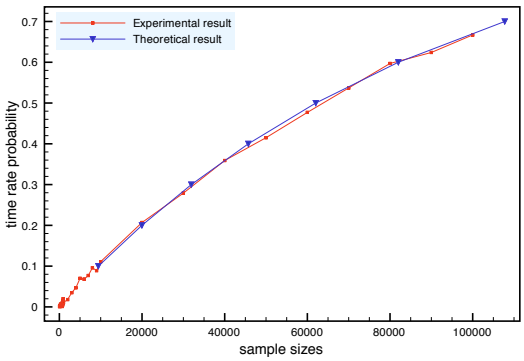

Figure 4. Probability to find a combination of sources with a transfer time equals to the optimal time, as function of the size of samples for $\mathcal{SG}_{11}$

To say that we have a better schedule, the sum of the mintime found and the time required to find it has to be lower than the average transmission time. To address this issue we execute the approximate algorithm with different sample sizes (one run by sample) and we measure the algorithm execution time. As expected, the execution time is proportional to the sample size. Using realistic numbers for the capacity of the links, relative to the size of data to be transmitted, we still save a significant time. For example with a sample size equal to $N = 600$, wa can have a transfer time inferior to $80s$ with a great probability, and an algorithm execution time equal to $15s$.

B. Grid'5000

In this section, we consider the grid5000 [11] architecture. We model it by a graph $\mathcal{GG} = (V, E)$ where $card(V) = 154$ and $card(E) = 182$. We have 62 requests, $card(\mathcal{S}) = card(R) = 62$, and each source has to transfer 50 GB. In such environment, $card\mathcal{P}(R) = 3.14699733(10^{85})$, it is intractable to use the exhaustive algorithm. So, we use the approximate algorithm with a sample of size 10^6. The lowest transfer time found is $T = 307s$

From the formula, $T = 307s$ is the smallest value of 97% of the possible schedules. Even if the time $T = 307$ is not the optimal one, it is obtained with a probability equal to 0.75 with a sample having size $N = 20$ (see Figure 5). This can be done in $17.95s$. So the global time is $324.95s$, which still lower than the average transmission time obtained with the sample of 10^6 combinations.

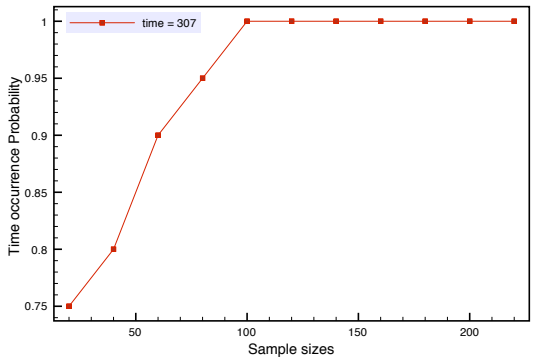

Figure 5. Probability to find a combination of sources with a transmission time equals to the optimal time, as function of the size of samples for $\mathcal{GG}$

VI. Conclusions

We have considered a bulk data transfer problem and especially a checkpoint images transfer. In the context of data Grid technology, this problem is considered as a very important issue. We have proposed a best effort scheduling to compute a communication plan when all communications that have to be scheduled are known at the initialization time.

To avoid the factorial combinatorial explosion of an exact and exhaustive algorithm, we have proposed an approximate algorithm based on the uniform random sampling of possible schedules. To study its effectiveness, we have implemented a simulator and carried out a set of experiments illustrating that even with a sample including a low number of combinations, the algorithm provides with high probability a transfer time much lower than the average transfer time.

References

[1] F. Bouabache, T. Herault, G. Fedak, and F. Cappello, "Hierarchical replication techniques to ensure checkpoint storage reliability in grid environment," in *CCGRID '08: Proceedings of the 2008 Eighth IEEE International Symposium on Cluster Computing and the Grid*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 475–483.

[2] R. Guillier, S. Soudan, and P. Vicat-Blanc Primet, "TCP variants and transfer time predictability in very high speed networks," in *Infocom 2007 High Speed Networks Workshop*, May 2007.

[3] L. Marchal, P. Primet, Y. Robert, and J. Zeng, "Optimizing network resource sharing in grids," in *IEEE Global Telecommunications Conference GlobeCom'2005*, 2005.

[4] H. Zhang, K. Keahey, and W. Allcock, "Providing data transfer with qos as agreement-based service," in *SCC '04: Proceedings of the 2004 IEEE International Conference on Services Computing*. Washington, DC, USA: IEEE Computer Society, 2004, pp. 344–353.

[5] L. Marchaland, P. V.-B. Primet, Y. Robert, and J. Zeng, "Optimal bandwidth sharing in grid environment." in *15th IEEE International Conference on High Performance Distributed Computing*, 2006.

[6] B. Chen and P. V.-B. Primet, "Scheduling deadline-constrained bulk data transfers to minimize network congestion," in *CCGRID*, 2007, pp. 410–417.

[7] K. Munir, P. Vicat-Blanc Primet, and M. Welzl, "Grid network dimensioning by modeling the deadline constrained bulk data transfers," in *11th IEEE International Conference on High Performance Computing and Communications (HPCC-09)*, Seoul, Korea, June 2009.

[8] S. Soudan, R. Guillier, and P. V.-B. Primet, "End-host based mechanisms for implementing flow scheduling in gridnetworks," in *GridNets*, 2007.

[9] J. Darbon, "Global optimization for first order markov random fields with submodular priors," *Discrete Applied Mathematics*, vol. 157, no. 16, pp. 3412 – 3423, 2009.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. Cambridge, MA: MIT Press, 2001.

[11] F. Cappello, E. Caron, M. Dayde, F. Desprez, Y. Jegou, P. V.-B. Primet, E. Jeannot, S. Lanteri, J. Leduc, N. Melab, G. Mornet, B. Quetier, and O. Richard, "Grid'5000: A large scale and highly reconfigurable grid experimental testbed," in *SC'05: Proc. The 6th IEEE/ACM International Workshop on Grid Computing CD*. Seattle, Washington, USA: IEEE/ACM, Nov. 2005, pp. 99–106.