



Laboratoire de l'Informatique du Parallélisme

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

***Simultaneous Scheduling of Replication
and Computation for Data-Intensive
Applications on the Grid***

Frédéric Desprez,
Antoine Vernois

January 2005

Research Report N° RR2005-01

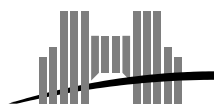
École Normale Supérieure de Lyon

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



INRIA



Simultaneous Scheduling of Replication and Computation for Data-Intensive Applications on the Grid

Frédéric Desprez, Antoine Vernois

January 2005

Abstract

One of the first motivations of using grids comes from applications managing large data sets like for example in High Energy Physic or Life Sciences. To improve the global throughput of software environments, replicas are usually put at wisely selected sites. Moreover, computation requests have to be scheduled among the available resources. To get the best performance, scheduling and data replication have to be tightly coupled which is not always the case in existing approaches.

This paper presents an algorithm that combines data management and scheduling at the same time using a steady-state approach. Our theoretical results are validated using simulation and logs from a large life science application (ACI GRID GriPPS).

Keywords: scheduling, replication, data management, grid computing

Résumé

L'une des principales motivations pour utiliser les grilles de calcul vient des applications utilisant de larges ensembles de données comme, par exemple, en Physique des Hautes Énergies ou en Science de la Vie. Pour améliorer le rendement global des environnements logiciels utilisés pour porter ces applications sur les grilles, des réplicats des données sont déposés sur différents sites sélectionnés. De plus, les requêtes de calcul doivent être ordonnancées en tenant compte des ressources disponibles. Pour obtenir de meilleures performances, l'ordonnancement des requêtes et la réplication des données doivent être étroitement couplés ce qui n'est généralement pas le cas dans les approches existantes.

Cet article présente un algorithme qui combine la gestion des données et l'ordonnancement en utilisant une approche en régime permanent. Nos résultats théoriques sont validés par simulations et par l'utilisation des traces d'un serveur de calcul d'application de Sciences de la Vie (ACI GRID GRIPPS).

Mots-clés: ordonnancement, réplication, gestion de données, grille de calcul

1 Introduction

One of the first motivations of using grids [6, 13] comes from the applications managing large data sets [10, 22] like for example in High Energy Physics [16] or Life Sciences [18]. To improve the global throughput of software environments, replicas are usually put at wisely selected sites. Moreover, computation requests have to be scheduled among the available resources. To get the best performance, scheduling and data replication have to be tightly coupled which is not always the case in existing approaches.

Usually, in existing grid computing environments, data replication and scheduling are two independent tasks. In some cases, replication managers are requested to find best replicas in term of access costs. But the choice of the best replica has to be done at the same time as the schedule of computation requests.

Our motivating example comes from an existing life science application (see Section 2). This kind of application has usually the following characteristics: a large number of independent tasks of small duration (searching for signature or functional site of a protein or protein family into a databank), reference databases from some MBs to several GBs which are updated on a daily or weekly basis, several computational servers available on the network, and the size of the overall data set is too important to be replicated on every computational server on the whole. The resolution of such application on the grid leads to solve two problems related to replication: **finding how (and where) to replicate the databases** and **choosing wisely the data to be deleted when new data have to be stored**. On the scheduling side, **computation requests must be scheduled on servers by minimizing some performance metric, taking into account the data location**.

This paper presents an algorithm that combines data management and scheduling simultaneously using a steady-state approach. Our theoretical results are validated using simulation and logs from a large life science application.

This paper is organized as follows. In a first section, we present the application that motivated this work. In Section 3, we discuss some previous work around data replication, web cache mapping, data and computation scheduling. In Section 4, we present our model of the problem and the algorithm we designed to solve it. Finally, before some conclusions and our future work, we discuss our experimentations using the OptorSim simulator [4] for replica managers.

2 Motivation Example

Our motivation for this work comes from an application in life science and more precisely around the search of sites and signatures of proteins into databanks of protein sequences.

Genomic acquiring programs such as full genomes sequencing projects are producing large amounts of data. Functional sites and signatures of protein are very useful for analyzing these data or for correlating different kind of existing biological data. Sites and signatures of protein can be expressed using the syntax defined by the PROSITE [7] databank, and written as a kind of regular expression. Then, the search of functional sites or signature into databanks can be very similar to simple pattern matching except that some biological relevant error between search pattern and matching protein can be allowed. These methods can be applied, for example, to identify and find a characterization of the potential functions of new sequenced proteins, or to clusterize the sequences contained into international databanks into

Number of databanks	23
Number of algorithms	8
Number of couple algorithm-databanks	80
Number of requests	88730
Average size of databanks	1 GB
Size of smallest databank	1 MB
Size of largest databank	12 GB

Table 1: Informations extracted from logs of an existing bioinformatic cluster.

families of proteins. Most of the time, this kind of analysis, i.e. searching for a matching protein into a databank, is quite fast and its execution time mainly depends on the size of the databanks, but the number of requests for such analysis can also be very high.

Figure 1 describes the classical architecture of a bioinformatic application. We can notice two kind of components connected together by the Internet network. On one side, there is a set of clients which submit requests to computational servers. Clients are seen as personal computers that have no knowledge from each others but which are often gathered in some big sites. Usually, these are office computer of researchers in biology or bioinformatics research centers. Computational servers are dedicated to computation. They usually are single processors computers or, sometime clusters of computers. These servers locally store a limited number of reference databanks and algorithms on which they can be applied. Often, they are independent from each other and are located and managed in bioinformatic centers. Client can access computational servers through web portals or directly asking for an account to servers administrators.

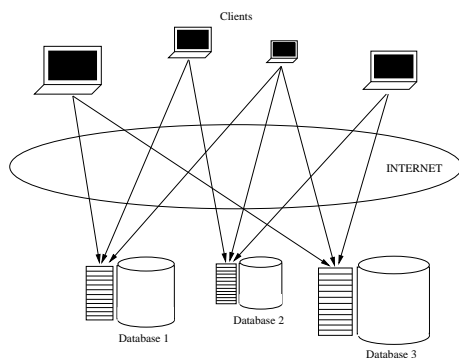


Figure 1: Current view of a Bioinformatic Application.

We accessed the logs of such a cluster that allow users to apply some well known algorithms to existing databanks. The cluster is located at IBCP [1] in Lyon (France), a research institute on biology and chemistry of proteins, and it is managed by the bioinformatic team of this laboratory. Input of such requests are user's protein sequences or signatures and usually do not exceed a few kilobytes. This is a centralized cluster with limited capacities so only major databanks and algorithms are available. Table 1 shows some information extracted from these logs.

3 Related Work

Data replication has attracted much attention over the last decade. Our work is connected to several others: high performance web caches, data replication, and scheduling in grids.

With the rapid growth of the Internet, scalability became in major issue for the design of high performance web services [26]. Several researches have studied how to optimally replace data in distributed web caches [8, 21]. Even if this problem seems to be close to ours, the fundamental difference between the two is that our problem has a non-negligible computation cost that depends upon the speed of the machine hosting a given replica.

In computation grids, some work exist around replication [19] and among them the researches for the Datagrid project from the CERN [2]. OptorSim [4, 11] allows to simulate data replication algorithms over a grid. This tool is more precisely described in Section 5.1. In [3], several strategies are simulated like unconditional replication (oldest file deleted, LRU) and with an economic approach. The target application is the data management of the Datagrid physic application. Simulation shows that the economical model is as fast as classical algorithms.

In [17], the authors describe Stork, a scheduler for data mapping in grid environments. Data are considered as resources that have to be managed as computation resources. This environment is mainly used to be able to map data close to computation during the scheduling of task graphs in Condor [25].

In [20], the scheduling of computation is linked to a previous data mapping. Tasks are scheduled on least loaded processors close to sites where data are stored. Replication is also used to improve performance.

The closest researches to the results presented in our paper are the one that aim to schedule computation requests and data mapping on remote sites at the same time. In [23, 24], several strategies are evaluated to manage data and computation scheduling. These strategies are either strongly related to the scheduling of computation or completely disconnected. However, these strategies are highly dynamic and the mapping is not proved close to the optimal. In [9], the authors present an algorithm (Integrated Replication and Scheduling Strategy) in which performance are iteratively improved by working alternatively on the data mapping and the task mapping.

4 Joint Data and Computation Scheduling Algorithm

In this section, we present the algorithm we designed that combines data replication and scheduling (Scheduling and Replication Algorithm or SRA).

Figure 2 shows one typical architecture used in our motivation application. Several clients are connected through the Internet to clusters managing databases and computation requests. Databases can be replicated on different servers. We also assume that a given server can host different databases at the same time provided that there is enough storage and the application related to this database is available on this server. Figure 3, present the model we use in our algorithm. We assume that a single broker manages data replication and scheduling of requests.

4.1 Hypothesis

We assume that we have the following:

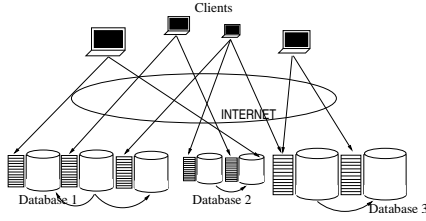


Figure 2: Data Replication and Load-Balancing of Computation Requests.

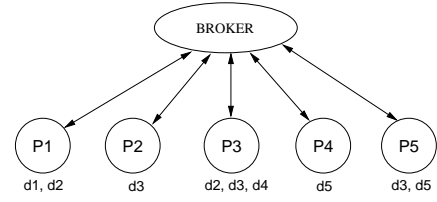


Figure 3: Theoretical Model Used in the SRA Algorithm.

- a set of computational servers P_i , $i \in [1..m]$.
- a set of data d_j , $j \in [1..n]$. Data d_j is of size $size_j$
- a set of algorithms a_k , $k \in [1..p]$ that use one d_j as an input.

We call a request, or task, $R_{k,j}$ a couple (a_k, d_j) where a_k is an algorithm and d_j is a data which will be used as an input of the algorithm a_k . All algorithms can not be applied on all kind of data, so we define $v_{k,j} = 1$ if $R_{k,j}$ is a request that is possible, otherwise $v_{k,j} = 0$. The complexity of algorithm a_k is linear in time with the size of the data. Thus the amount of computation needed to compute a request $R_{k,j}$ is $\alpha_k \cdot size_j + c_k$, where α_k and c_k are two constants defined for each algorithm a_k . For each server, we also introduce $n_i(k, j)$, which is the number of requests $R_{k,j}$ that will be executed on server P_i .

A server P_i is described by two constants: its computational power w_i and its storage capacity m_i .

$f_{k,j}$ is the fraction of request of type $R_{k,j}$ in the pool of requests. We suppose that this proportion of request is always the same whatever the interval of time you consider as soon as it is large enough.

Our study focus on managing data and their replication taking all these parameters into account to improve the computation time of a set of requests. We also make the assumption that for each data, there is at least one server with enough space to store it.

From data point of view, there are two possibilities for the platform:

- the total space available is large enough to store, at least, one replica of each data.

$$\sum_{i=1}^m m_i \geq \sum_{j=1}^n size_j \quad (1)$$

- the total space available is too small, and only a subset of data can be stored in the platform at a given time.

$$\sum_{i=1}^m m_i < \sum_{j=1}^n size_j \quad (2)$$

In the following, we will consider that we are in the first case when for each data there is at least one server that can hold a replica of it. We will try to find the best way to place data in these conditions.

As we have enough space, we want that there is at least one replica of each databanks in the system. Let $\delta_i^j = 1$ if there is a replica of data d_j on server P_i , $\delta_i^j = 0$ otherwise.

$$\forall j \sum_{i=1}^m \delta_i^j \geq 1 \quad (3)$$

But space on each server is limited by its storage capacity. So total size of data stored on server P_i cannot exceed m_i

$$\forall i \sum_{j=1}^n \delta_i^j \cdot size_j \leq m_i \quad (4)$$

The number of requests a server P_i can handle is restricted by its computation capacity. Thus the amount of computation that a server will execute cannot exceed w_i .

$$\forall i \sum_{k=1}^p \sum_{j=1}^n n_i(k, j) (\alpha_k \cdot size_j + c_k) \leq w_i \quad (5)$$

To compute a request $R_{k,j}$ on server P_i , the data d_j should be stored on this server. If it is not the case, then $n_i(k, j)$ should be equal to 0, otherwise, $n_i(k, j)$ is limited by the maximal number of requests $R_{k,j}$ this server can handle.

$$\forall i \forall j \forall k n_i(k, j) \leq v_{k,j} \cdot \delta_i^j \cdot \frac{w_i}{\alpha_k \cdot size_j + c_k} \quad (6)$$

Let TP be the throughput of the platform, *i.e.*, TP is the number of requests that can be executed per unit time on the platform. The ratio of each kind of request is defined by $f(j, k)$. So, the number of requests of type $R(k, j)$ that is executed is restricted by this ratio in order to avoid to take in account more requests than the number that will be submitted.

$$\forall j \forall k \sum_{i=1}^m n_i(k, j) = f_{k,j} \cdot TP \quad (7)$$

Considering all the previous equations, our goal is to define a mapping, *i.e.* δ_i^j , of data to be able to compute as much as possible requests in a fixed period of time. It means that we want to maximise the throughput TP . From previous constraints, and this goal, we can define the following linear program.

linear programming formulation

MAXIMIZE TP ,

SUBJECT TO

$$\left\{ \begin{array}{ll} (1) & \sum_{j=1}^n \delta_i^j \geq 1 & 1 \leq i \leq m \\ (2) & \sum_{j=1}^n \delta_i^j \cdot size_j \leq m_i & 1 \leq i \leq m \\ (3) & n_i(k, j) \leq v_{k,j} \cdot \delta_i^j \cdot \frac{w_i}{\alpha_k \cdot size_j + c_k} & 1 \leq i \leq m, 1 \leq j \leq n, 1 \leq k \leq p \\ (4) & \sum_{k=1}^p \sum_{j=1}^n n_i(k, j) (\alpha_k \cdot size_j + c_k) \leq w_i & 1 \leq i \leq m \\ (5) & \sum_{i=1}^m n_i(k, j) = f_{k,j} \cdot TP & 1 \leq i \leq m, 1 \leq j \leq n \\ (6) & \delta_i^j \in \{0, 1\} & 1 \leq i \leq m, 1 \leq j \leq n \end{array} \right.$$

The solution of this linear program will give us a placement for the databanks on the servers but also, for each kind of job, on which server they should be executed. More precisely, for a kind of request $R_{k,j}$, we know how many job can be executed on the platform and we also know how many requests of this kind should be executed on each server to reach optimal throughput. Thus, with the placement of data, the linear program also gives good information for the scheduling of requests.

4.2 NP-Completeness Proof

Definition 1 (SCHEDULE-REPLICATION-DEC). *Given m processors P_i with computation power w_i and storage m_i , n data d_j of size $size_j$, p algorithms a_k that can be applied on data d_j , $v_{k,j}$ defining if algorithm a_k can be applied on d_j , α_k and c_k such that $\alpha_k \cdot size_j + c_k$ is the amount of computation needed to compute algorithm a_k on data d_j , $f_{k,j}$ the proportion of requests of a_k on d_j , and a bound K , is it possible to find a placement δ_i^j and a scheduling $n_i(k, j)$ such that the throughput $TP \geq K$?*

Theorem 1. *SCHEDULE-REPLICATION-DEC is NP-complete.*

Proof. First SCHEDULE-REPLICATION-DEC obviously belong to the class NP. To prove its NP-completeness, we proceed by a reduction from 2-PARTITION, which is known to be NP-complete [14]. An arbitrary instance \mathcal{I}_1 of 2-PARTITION is the following: given N positive integer S $a_i, 1 \leq i \leq N$ and a positive integer S such that $2S = \sum_i a_i$, is it possible to find a set $I \subset \{1..N\}$ such that $\sum_{i \in I} a_i = \sum_{i \notin I} a_i = S$? We construct the following instance \mathcal{I}_2 of SCHEDULE-REPLICATION-DEC: let N databanks of $size_j = a_j$, P_1 and P_2 be two processors such that $w_1 = w_2 = S + N$ and $m_1 = m_2 = S$, one algorithm a_1 with parameter $\alpha_1 = 1$ and $c_1 = 1$, $f_{1,j} = \frac{1}{N}$, $v_{1,j} = 1$ and $K = N$. The construction of such instance \mathcal{I}_2 from \mathcal{I}_1 is polynomial. We show that \mathcal{I}_2 admits a solution if and only if \mathcal{I}_1 does:

- Assume first that \mathcal{I}_1 admits a solution and let I be a subset of $\{1..N\}$. For each $j \in I$, we define $\delta_1^j = 1$, $\delta_2^j = 0$, $n_1(1, j) = 1$ and $n_2(1, j) = 0$. For each $j \notin I$, we define $\delta_1^j = 0$, $\delta_2^j = 1$, $n_1(1, j) = 0$ and $n_2(1, j) = 1$. With such a placement and scheduling, we have all databases placed at least on one processor. The size of all data stored on processor P_1 is

$$\sum_{j=1}^N \delta_1^j \cdot size_j = \sum_{j \in I} size_j = \sum_{j \in I} a_j = S$$

The amount of computation on processor P_1 is

$$\sum_{j=1}^N n_1(1, j)(\alpha_k \cdot size_j + c_k) = \sum_{j \in I} (a_j + 1) = S + \sum_{j \in I} 1 \leq S + N$$

The same results are obtained for processor P_2 . The throughput of the platform is equal to

$$TP = \sum_{i=1}^2 \sum_{j=1}^N n_i(1, j) = \sum_{j=1}^N n_1(1, j) + \sum_{j=1}^N n_2(1, j) = \sum_{j \in I} 1 + \sum_{j \notin I} 1 \geq N$$

Therefore $TP \geq N$ and \mathcal{I}_2 has a solution.

- Assume now that \mathcal{I}_2 has a solution, and let δ_i^j and $n_i(k, j)$ be the placement and scheduling of this solution. On each processor, the space of stored databanks does not exceed m_i , then

$$\forall i \in \{1, 2\} \quad \sum_{j=1}^N \delta_i^j \cdot size_j \leq m_i.$$

But, $size_j = a_j$, $m_i = S$ and $\sum_j a_j = 2S$ then, if we add the two previous inequations:

$$\sum_{j=1}^N (\delta_1^j + \delta_2^j) \cdot a_j \leq 2S$$

$$\sum_{j=1}^N (\delta_1^j + \delta_2^j) \cdot a_j \leq \sum_{j=1}^N a_j$$

But $\delta_i^j \in \{0, 1\}$ and as each data is stored at least one time in the platform we have

$$\forall j \in \{1..N\} \quad \delta_1^j + \delta_2^j \geq 1$$

so $\forall j \in \{1..N\}$ either $\delta_1^j = 1$ and $\delta_2^j = 0$ either $\delta_1^j = 0$ and $\delta_2^j = 1$. Then let $I = \{j, \delta_1^j = 1\}$, and we have

$$\sum_{j \in I} a_j = \sum_{j=1}^N \delta_1^j \cdot a_j = S$$

and \mathcal{I}_1 has a solution. ■

4.3 Integer Solution Approximation

As δ_i^j are equal to 0 or 1, this linear program is mixed, both rational and integer. As proved in previous section, this problem is NP-complete. So we choose to solve the relaxed program over rational number. Starting from the rational solution, we will find an approximate integer solution for the problem. We try to approximate the solution using the known information.

Algorithm 1 is used to find an integer approximation of the relaxed problem. We start from the relaxed solution of the linear problem and construct three sets: S_0 will contain couples (i, j) where $\delta_i^j = 0$ in the solution, S_1 contains (i, j) where $\delta_i^j = 1$ and S all (i, j) that are not yet in S_1 nor in S_0 . Then we compute *notPlaced*, the set of databanks indexes that are not yet placed anywhere. If this set is not empty *i.e.*, there is still at least one databank that is not mapped, we choose a (i_1, j_1) with $j_1 \in notPlaced$ for which the sum of the computation time required for each algorithm that can be applied to the concerned databanks multiplied by the current value of δ_i^j is the highest. If *notPlaced* is empty, we choose a couple (i_1, j_1) from S with the same method. We add this couple to S_1 . For each couple in S_1 we then add the constraint $\delta_i^j = 1$ to the original relaxed linear program. We do the same with constraint $\delta_i^j = 0$ for couple in S_0 . We then solve the new linear program and restart to the beginning until $S = \emptyset$. If with the construction, the linear program becomes infeasible, we remove the chosen (i_1, j_1) from S_1 , add it to S_0 and reconstruct the new linear program.

Algorithm 1 Integer Approximation algorithm

```

1: Solve relaxed linear program  $lp$ 
2: let  $S_0 = \{(i, j) | \delta_i^j = 0\}$ 
3: let  $S_1 = \{(i, j) | \delta_i^j = 1\}$ 
4: let  $S = \{(i, j) | (i, j) \notin S_0 \text{ and } (i, j) \notin S_1\}$ 
5: let  $notPlaced = \{j \in [1..n] | \forall i \in [1..m] \delta_i^j \neq 1\}$ 
6: if  $notPlaced \neq \emptyset$  then
7:    $(i_1, j_1) | (\sum_{k=1}^p m_{j_1} \cdot \alpha_k + c_k) \cdot \delta_{i_1}^{j_1} = \max_{\{j \in notPlaced, (i, j) \in S\}} \sum_{k=1}^p m_j \cdot \alpha_k + c_k) \cdot \delta_i^j$ 
8:   add  $(i_1, j_1)$  to  $S_1$ 
9: else
10:   $(i_1, j_1) | (\sum_{k=1}^p m_{j_1} \cdot \alpha_k + c_k) \cdot \delta_{i_1}^{j_1} = \max_{\{(i, j) \in S\}} \sum_{k=1}^p m_j \cdot \alpha_k + c_k) \cdot \delta_i^j$ 
11:  add  $(i_1, j_1)$  to  $S_1$ 
12: end if
13: for all  $\delta_i^j \in S_1$  do
14:   add constraint  $\delta_i^j = 1$  to  $lp$ 
15: end for
16: for all  $\delta_i^j \in S_0$  do
17:   add constraint  $\delta_i^j = 0$  to  $lp$ 
18: end for
19: Solve new  $lp$ 
20: if  $lp$  is not feasible then
21:   Remove  $(i_1, j_1)$  from  $S_1$ 
22:   Add  $(i_1, j_1)$  to  $S_0$ 
23:   Redo step 13 to 18 and solve it again
24: end if
25: Go to step 2 until  $S = \emptyset$ 

```

4.4 A Greedy Solution

Starting with the same platform and algorithm models, we also design the greedy algorithm (Algorithm 2) to solve the mapping problem. The idea behind this algorithm is to try to map data that need the most computational power to the server that has the most computation capacities first.

The algorithm starts by computing the amount of computation needed by each data proportionally to its usage. Then we sort data by decreasing values of this amount. We also sort the list of servers by decreasing computation abilities. We try to map the data that need the most computational power to the server that has the highest computation capacity. If there is not enough space, we try to map the data on the second server and so on and so forth until the data is mapped. Then, we try to place the second data by computation need to the first server. We do this operation for each data. If a data cannot be placed on any server we skip it and try to place the following data item. We restart from the beginning of data list till there is enough space available to place a data on the platform.

Algorithm 2 Greedy algorithm for mapping problem

```

1: for all data  $d_j$  do
2:    $s_j = \sum_{k=1}^p f(j, k) \cdot (\alpha_k \cdot m_j + c_k)$ 
3: end for
4: Sort data by decreasing values of  $s_j$  in sortData
5: Sort servers by decreasing values of  $w_i$  in sortServer
6: while No data have been mapped do
7:   for  $j = 0; j < n; j++$  do
8:      $i = 0, place = false$ 
9:     while  $!place$  and  $i < n$  do
10:      if there is enough space on server  $sortServer[i]$  for data  $sortData[j]$  then
11:        if data  $sortData[j]$  not already on server  $sortServer[i]$  then
12:          map this data on that server
13:           $place = true$ 
14:        end if
15:         $i++$ 
16:      end if
17:    end while
18:  end for
19: end while

```

5 Experiments

To experiment the results of our model, we used OptorSim [4, 11], a simulator of Data Grid environments developed in the Work Package 2 of EU Datagrid project [2]. We have modified OptorSim to exactly match our needs.

This simulator takes a Grid topology and configuration file as an input that let you define its behaviour and all input parameters.

5.1 Experimental Environment

The simulated grids have five major components. Computing Elements (CE) act like gateways, or masters of a batch scheduler system and will distribute jobs that are submitted to them to their Worker Nodes (WN). Worker Nodes execute jobs and are defined by their computation power expressed in flops. All Worker Nodes managed by the same CE have the same capacity of computation, but WN from different CE may have different capacities. So we have an heterogeneous cluster of homogeneous nodes, which is usually the case on real grids.

The third kind of component is the Storage Element (SE). It is where data are stored and is defined by its storage capacity (in MB). A same file can be stored on different SE at the same time, and each SE can decide to delete a data from its storage if this file is not used and space is required for another file. As a SE does not ask to other SEs or the Replica Manager (see below) if it can delete a file, it is possible that all copies of a same file are deleted. To prevent that, for each file, there is one master copy that cannot be deleted. The way master copies are distributed is defined in configuration file and can be random, following information that tell on which server to place each master copy or a list of SE where master copies will be

evenly distributed. This distribution is done before the beginning of the simulation. Thus, the cost of placing master copies is considered to be null. To work properly, a CE should have a local SE that is accessible by all of its Worker Nodes. Access time to data located on the local SE by a WN is considered to be null.

The Replica Manager (RM) has in charge all data movements between sites. And finally, jobs are created and scheduled by a unique Resource Broker which is able to instantiate communications with CE and RM to get all information needed about SE such as network bandwidth, job queues, etc. for scheduling purpose. The way jobs are created and scheduled is defined in the configuration file. In our case, a job is defined by an algorithm and a databank on which the algorithm is applied. Jobs are created following the information given in a separate file consisting on a succession of couples (algorithm, data). Such file is generated from logs of existing bioinformatics clusters.

For our experiments, we extracted from raw logs, all information about data sets and algorithm usage. With external information about data sizes, algorithm computation costs, and a description of the target platform, we generated the concrete instance of the linear program described in Section 4. This linear program is solved using *lp_solve* [5]. The results give us all information about data mapping and job scheduling. These outputs are used, with other configuration files, as inputs for the simulator.

For the experiments, the topology of the simulated platform is given in Figure 4 and this topology is inspired from the architecture of the European DataGrid testbed. There are 10 CE with associated SE and 7 routers without any storage nor computation abilities.

Requests are submitted to the RB with a frequency around ten per second. This could seem to be a very high rate, but discussions with the biologist and bioinformatic community lead to the conclusion that the more computation power we can give them, the more they will use.

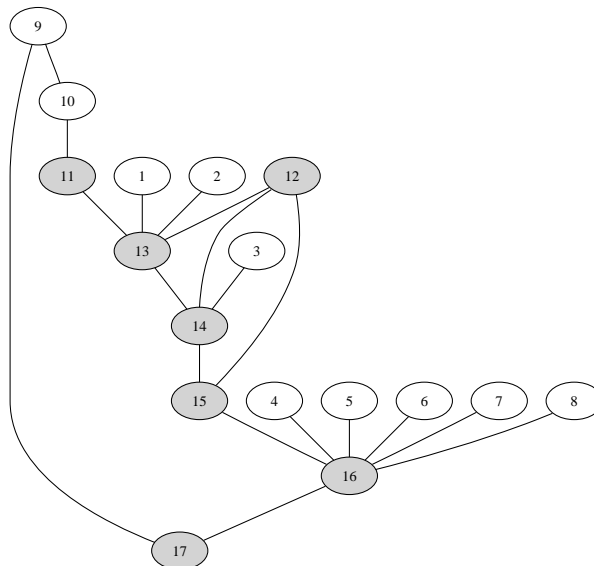


Figure 4: Topology of the simulated platform. Nodes 1 to 10 are CE with SE, nodes 11 to 17 (gray filled) are routers without any computation or storage abilities.

5.2 Experiment Results and Discussion

In this section we will discuss our experiments using OptorSim and the results we obtained. We have done simulations for three kinds of mapping and schedulers.

The first one, *SRA*, corresponds to our algorithm. Scheduling and mapping that are used for the simulation are those that match the solution of our linear program.

In the *MCT*¹ simulation, only the mapping has been done using the results of the linear program. The scheduling is on-line: at each request submission, it tries to find the computation server that should be able to finish this task first (considering time to retrieve data if needed and computation time of all jobs already scheduled on the CE).

Finally, the *greedy* simulation is done using the mapping of the greedy algorithm. The scheduling is done with the previous on-line scheduler. Simulations have been done for a pool a 40000 requests.

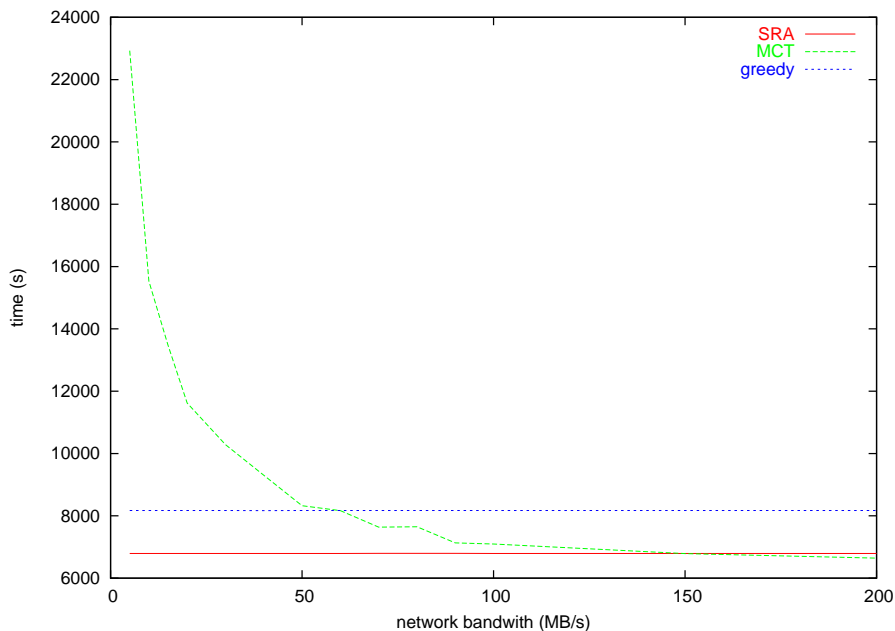


Figure 5: Execution time for 40000 jobs as function of the network bandwidth between CE for different mapping and scheduling algorithms.

Figure 5 shows the execution time of whole set of requests depending on the network bandwidth. In this simulation, the bandwidth between nodes is chosen to be homogeneous to see more easily its impact on execution time. Figure 6 represents the volume of data transferred during the simulation for each kind of placement and scheduling. On Figure 5, we can see that for SRA and greedy, the time of execution is totally constant and independent of network bandwidth. It is due to the fact that there are no data movement with these two methods as we can see in Figure 6.

But reasons for which there are no movement are not the same in both cases. With SRA algorithm, the scheduling is computed at the same time as the placement. So the scheduler always schedules a job on a server that has needed data for this request. In the greedy case,

¹Minimum Completion Time

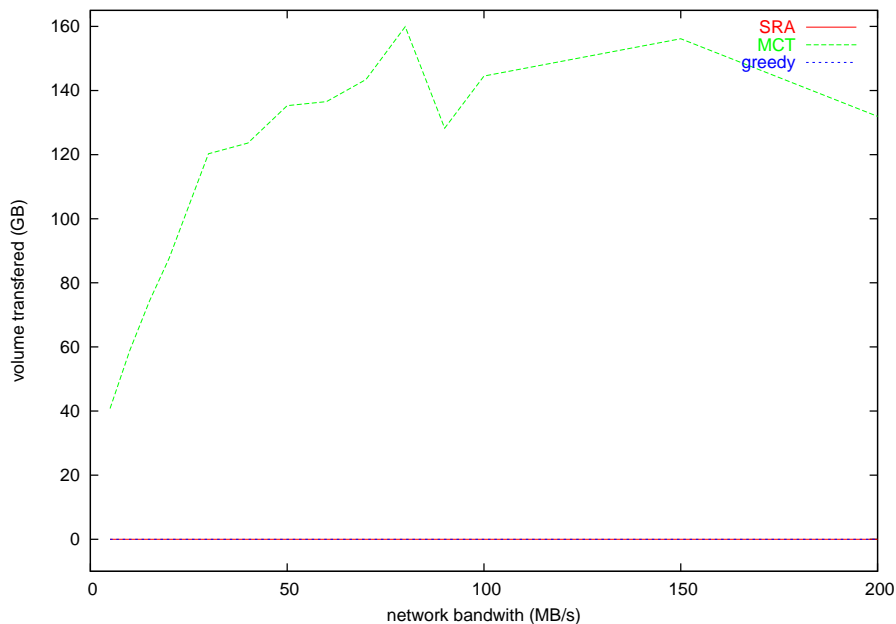


Figure 6: Volume of data transferred during execution.

the scheduling uses an on-line MCT method but there is still no data movement because the algorithm totally fills the available space in the platform. So the MCT scheduler always schedules requests where a data is available. As shown by Figure 6, MCT does a lot of transfers of data favouring the execution time of current request to schedule. But its lack of knowledge on request usage scheme leads him to perform a lot of errors and useless data transfers. Then, it becomes efficient only when transfers costs are negligible in front of computation costs.

Figure 7 shows the execution time of same set of 40000 requests depending on the storage space available on the platform. The space is expressed as the ratio between the total volume of databanks and the global space available. For this simulation the network bandwidth is equal to 10MB/s.

We can notice that for all kind of mapping and scheduling algorithms, the execution time decreases with the increase of available space. It can be easily explained by the fact that the more space is available, the more replicas can be placed on different servers. As we can expect, when storage space is small, less than 8 times the size of databanks, our solution gives better results than greedy and MCT. The linear program makes a better use of restricted resources. With the increase of available space, the results of the greedy algorithm improves regularly to become better than the SRA algorithm. This appears when next to all databanks can be stored on each server.

Figure 8 is a zoom of the previous figure restricted to SRA and greedy simulations. When space storage is very limited, the results of our algorithm are not regular. That comes from our heuristic that constructs an integer solution of the linear program from the solution over rational numbers. With a small storage space, the impact of a bad mapping choice has a high impact on the objective function. In this case, we notice very high differences between value of the objective function of the approximation integer solution and the solution in rational numbers. When available space becomes large enough, our integer approximation gives the

same result of the objective value than resolution in rational number.

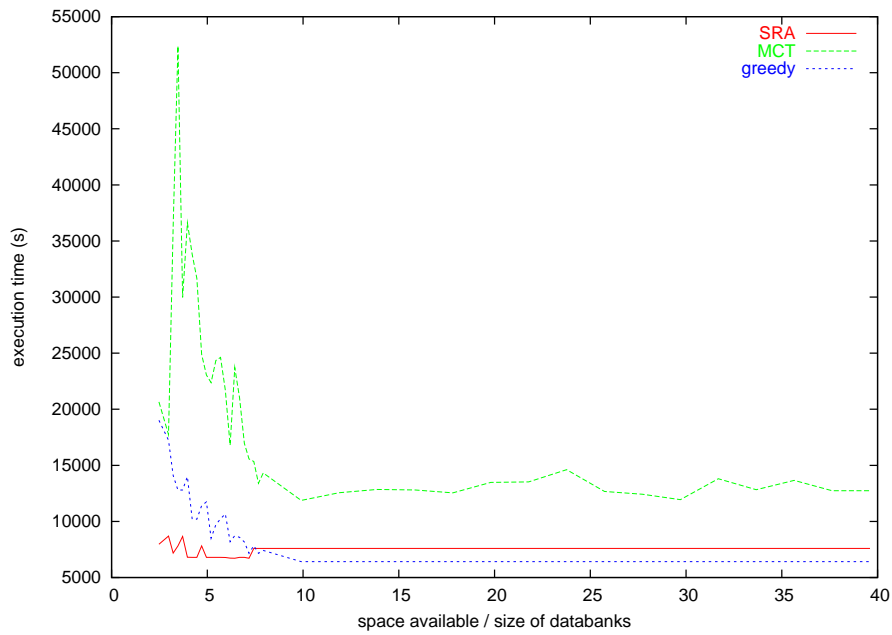


Figure 7: Execution time for 40000 jobs as a function of available space on SE.

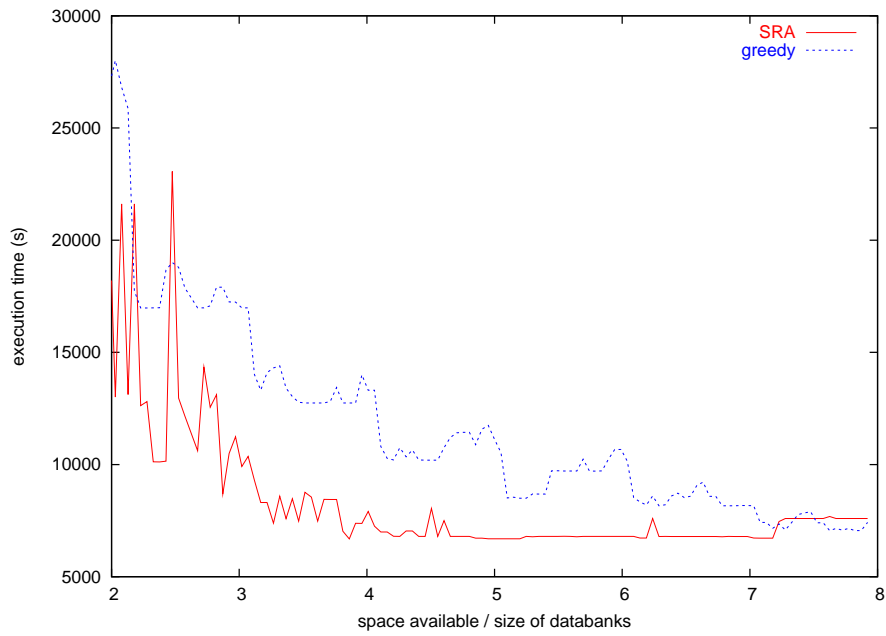


Figure 8: Zoom of execution time.

5.3 Comparison of Integer Approximation Algorithms

In order to obtain an estimation of the performance of our algorithm of integer approximation, we have compared it with a random method. These algorithms are very close, and the main difference between both is in the choice of the variable whose values will be approximated at each turn and the value that will be set. In the random method, the unknown that will be set is randomly choose between all δ_i^j that has not yet been set. Let call $\delta_{i_1}^{j_1}$ such a chosen unknown to be set. If $\delta_{i_1}^{j_1}$ is greater or equal to 0.5 in the previous solution of the linear program, then $\delta_{i_1}^{j_1}$ is set to 1 else, it set to 0.

Figure 9 shows the execution time of 40000 request for both approximation methods. As we can expect, our approximation algorithm gives better results than the random algorithm. That can be explained by the choose of the unknown set at each run of the linear solver. The method used in SRA takes into account information about databank’s usage, and let the final solution to stay close to the solution over rational number.

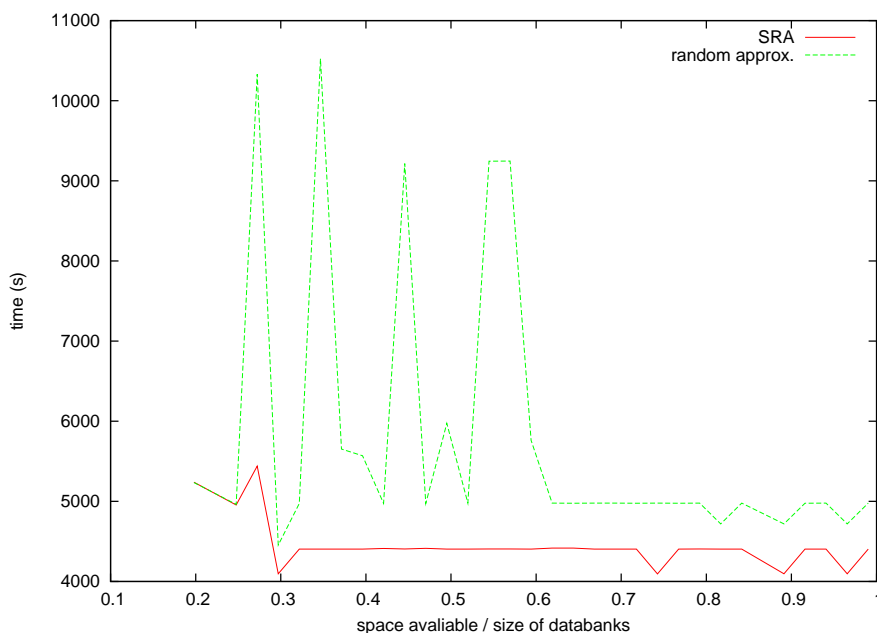


Figure 9: Execution time for 40000 jobs as a function of the network bandwidth between CE for different approximation algorithms.

6 Conclusion and Future Work

In this paper, we have presented an algorithm that computes at the same time the mapping of data and computational requests on these data.

Our approach uses a good knowledge of databank usage scheme and of the target platform. Starting with these information, we have designed a linear program and a method to obtain a mixed solution, *i.e.*, integer and rational numbers, of this program. With the OptorSim simulator, we have been able to compare the results of our algorithm to other approaches: a greedy algorithm for data mapping, and an on-line algorithm for the scheduling of requests.

We came to the conclusion that when the storage space available on the grid is not large enough to store all databanks that lead to very time consuming requests on all computation servers, then our approach improves the throughput of the platform. But our heuristic for approximating an integer solution of the linear program does not always give the best mapping of data and can give results that are very far from the value of the objective function in the solution over rational number.

Our future works will consist on adding communication costs for the requests in the model to be able to consider other kind of applications. We are also working on an implementation of these algorithm in the DIET [12] environment to deploy efficiently the GriPPS [15] application. A replica manager will be designed and developed in this environment.

Acknowledgements

The authors would like to thanks Christophe Blanchet for having inspired this work, his help with bioinformatic applications, and for the execution logs of the NPS@ server. We also would like to thanks Arnaud Legrand, Loris Marchal and Yves Robert for their work on steady-state scheduling and discussion about the model used in this article. The authors would also give a special thanks to Loris Marchal for his advices about the NP-completeness proof.

References

- [1] Institut de Biologie et Chimie des Protéines. <http://www.ibcp.fr>.
- [2] The European DataGrid Project. <http://www.eu-datagrid.org>.
- [3] W.H. Bell, D.G. Cameron, L. Capozza, A.P. Millar, K. Stockinger, and F. Zini. Simulation of Dynamic Grid Replication Strategies in OptorSim. In *Proc. of the 3rd Int'l. IEEE Workshop on Grid Computing (Grid'2002)*, Lecture Notes in Computer Science, Baltimore, USA, November 2002. Springer Verlag.
- [4] W.H. Bell, D.G. Cameron, L. Capozza, A.P. Millar, K. Stockinger, and F. Zini. OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies. *International Journal of High Performance Computing Applications*, 17(4), 2003. <http://edg-wp2.web.cern.ch/edg-wp2/publications.html>.
- [5] M. Berkelaar. LP_SOLVE. <http://www.cs.sunysb.edu/~algorithm/implement/lpsolve/implement.shtml>.
- [6] F. Berman, G.C. Fox, and A.J.H. Hey, editors. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley, 2003.
- [7] P. Bucher and A. Bairoch. A Generalized Profile Syntax for Biomolecular Sequences Motifs and Its Function in Automatic Sequence Interpretation. In R. Altman, D. Brutlag, P. Karp, R. Lathrop, and D. Searls, editors, *Proceedings 2nd International Conference on Intelligent Systems for Molecular Biology*, volume 2, pages 53–61. AAAIPress, 1994.
- [8] V. Cardellini, E. Casalicchio, M. Colajanni, and P.S. Su. The State of the Art in Locally Distributed Web-Server Systems. *ACM Computing Surveys*, 34(2):263–311, June 2002.

- [9] A. Chakrabarti, R.A. Dheepak, and S. Sengupta. Integration of Scheduling and Replication in Data Grids. In L. Bougé and V. K. Prasanna, editors, *Proceedings 11th International Conference on High Performance Computing (HiPC 2004)*, pages 375–385, Bangalore, India, December 2004. Springer.
- [10] A. Chervenak, I. Foster, C. Kesselman, C. Salisbury, and S. Tuecke. The Data Grid: Towards an Architecture for the Distributed Management and Analysis of Large Scientific Datasets. *Journal of Network and Computer Applications*, 23:187–200, 2001.
- [11] R. Carvajal-Schiaffino D.G. Cameron, A.P. Millar, C. Nicholson, K. Stockinger, and F. Zini. Evaluating Scheduling and Replica Optimisation Strategies in OptorSim. In *4th International Workshop on Grid Computing (Grid2003)*, Phoenix, Arizona, November 2003. IEEE Computer Society Press.
- [12] DIET. <http://graal.ens-lyon.fr/DIET/>.
- [13] I. Foster and C. Kesselman, editors. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2004.
- [14] M. R. Garey and D. S. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.
- [15] GRIPPS. <http://gripps.ibcp.fr/index.php>.
- [16] W. Hoscheck, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger. Data Management in an International Data Grid Project. In *First IEEE/ACM Int'l Workshop on Grid Computing (Grid 2000)*, December 2000.
- [17] T. Kosar and M. Livny. Stork: Making Data Placement a First Class Citizen in the Grid. In *Proceedings of 24th IEEE Int. Conference on Distributed Computing Systems (ICDCS2004)*, Tokyo, Japan, March 2004.
- [18] A. Krishnan. A Survey of Life Sciences Applications on the Grid. *New Generation Computing*, 22:111–126, 2004.
- [19] H. Lamahamedi, B. Szymanski, Zujun Shentu, and E. Deelman. Data Replication Strategies in Grid Environments. In *Proc. 5th International Conference on Algorithms and Architecture for Parallel Processing, ICA3PP'2002*, pages 378–383, Beijing, China, October 2002. IEEE Computer Science Press.
- [20] H.H. Mohamed and D.H.J Epema. An Evaluation of the Close-to-Files Processor and Data Co-Allocation Policy in Multiclusters. In *Cluster 2004*, pages 287–298. IEEE Computer Society Press, 2004.
- [21] S. Podlipding and L. Böszörményi. A Survey of Web Cache Replacement Strategies. *ACM Computing Surveys*, 35(4):374–398, December 2003.
- [22] X. Qin and H. Jiang. Data Grid: Supporting Data-Intensive Applications in Wide-Area Networks. Technical Report TR-03-05-01, University of Nebraska-Lincoln, Lincoln, NE, USA, May 2003.

- [23] K. Ranganathan and I. Foster. Decoupling Computation and Data Scheduling in Distributed Data Intensive Applications. In *Proceedings of the 11th International Symposium for High Performance Distributed Computing (HPDC-11)*, Edinburgh, July 2002.
- [24] K. Ranganathan and I. Foster. Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids. *Journal of Grid Computing*, 1(1):53–62, 2003.
- [25] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The condor experience. *Concurrency and Computation: Practice and Experience*, 2004.
- [26] C.Z. Xu, H. Jin, and P.K Srimani. Special Issue on Scalable Web Services and Architecture. *Journal on Parallel and Distributed Computing*, 63, 2003.