Semi-Static Algorithms for Data Replication and Scheduling Over the Grid

Frédéric Desprez

INRIA - GRAAL Project/LIP Laboratory
UMR CNRS, ENS Lyon, INRIA, UCBL 5668
F-69364 Lyon Cedex 07
Frederic.Desprez@ens-lyon.fr

Antoine Vernois

LBP Laboratory / PMC Team
UMR 6023 CNRS U. B. Pascal Clermont-Ferrand II
F-63177 Aubière cedex
Antoine.Vernois@univ-bpclermont.fr

***Abstract*— Managing large datasets has become one major application of grids. Life science applications usually manage large databases that should be replicated to let applications scale. The growing number of users and the simple access to Internet-based applications has stressed grid middlewares. Such environments are thus asked to manage data and schedule computation tasks at the same time. These two important operations have to be tightly coupled to get the best results.**

This paper presents several heuristics that combines data management and scheduling using a steady-state approach. Starting from an intial distribution given by the resolution of a linear program, we optimize the results at run-time to improve the quality of the data mapping and the scheduling of computation requests when the request distribution evolves. Our theoretical results are validated using simulation and logs from a large life science application.

I. Introduction

An early and important usage of grid environments [1], [2] comes from applications managing large data sets [3], [4] in fields such as High Energy Physics [5] or life sciences [6]. To improve the global throughput of software environments, replicas are usually put at carefully 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. In existing grid computing environments, data replication and scheduling are usually two independent tasks. In some cases, replication managers are used to find the best replicas in terms of access costs. However the choice of the best replica should be done at the same time as the scheduling of computation requests.

Our motivating example comes from an existing life science application. The application tasks involve searching for the signature or functional site of a protein or protein family in a database. They have the following characteristics: a large number of independent tasks of small duration and reference database sizes from several 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 large to be completely replicated on every computational server. In order to run such applications on the grid, one has 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 that takes into account the data location.

In a previous paper [7], [8], we have presented a static algorithm that provides simulataneous data management and scheduling using a steady-state approach. Using a model of the platform, the number of requests as well as their distribution, and the number and size of databases, we define a linear program to satisfy the constraints at every level of the platform in steady-state. The solution of this linear program will give us a placement for the databases on the servers as well as providing, for each kind of job, the server on which they should be executed. However, even if this algorithm is very efficient when the requests frequency and the load of the platform does not change over time, it reaches its limits rapidly on a highly dynamic environment.

Thus we have improved our algorithm by adding heuristics that allow some data to be deleted or moved on some specific computing servers and adapt scheduling policies. On the replication side, we need to choose which data should be more replicated, which data must be deleted to gain storage space. On the scheduling side, servers must be chosen according to their load at a given time. Thus this leaves us with different combinations of data and server management. This paper presents a study of the behavior of this semi-static algorithm under dynamic constraints and some solutions to keep the overall system well balanced. Our theoretical results are validated using simulation and logs from a large life science application.

This paper is organized as follows. In the first section, we discuss some previous work in the areas of data replication, web cache mapping, and data and computation scheduling. In Section III, we present our model of the problem and the algorithm we designed to solve it . In Section IV, we present our semi-static algorithm which improves the data mapping when requests frequencies change. Finally, before some conclusions and our future work, we discuss our experiments using the OptorSim simulator [9] for replica managers.

II. 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 a major issue for the design of high performance web services [10]. Several researchers have studied how to optimally replace data in distributed web caches [11], [12]. Although 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 are focused on replication [13]; an example is Datagrid project from the CERN [14]. OptorSim [9], [15], built within this project, allows one to simulate data replication algorithms over a grid. A modified version of this tool was used in our paper. In [16], several strategies are simulated including unconditional replication (oldest file deleted, LRU) and an economic approach. The target application is data management for the Datagrid physics application. Simulation shows that the economic 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 map data close to computations during the scheduling of task graphs in Condor [18].

In [19], the scheduling of computations is linked to a previous data mapping. Tasks are scheduled on the least loaded processors close to sites where data are stored. Replication is also used to improve performance. The research that is closest to the results presented in our paper seeks to schedule computation requests and data mapping on remote sites at the same time. In [20], [21] several strategies are evaluated to manage data and computation scheduling. These strategies are either strongly related to the scheduling of computation or completely disconnected. In [22], the authors present an algorithm (Integrated Replication and Scheduling Strategy) in which performance is iteratively improved by working alternatively on the data mapping and the task mapping. However, the approach still separates replication and scheduling, but in a connected fashion.

III. Static Joint Replication and Scheduling Algorithm

In this section, we present the first algorithm we designed [7], [8]. It uses a modelization of the target platform (processor speed, storage capacity, size of databases, frequency of requests). We called this algorithm SRA for Scheduling and Replication Algorithm.

Typical architecture used in our motivating application is made of several clients 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 that the applications using these databases are available on this server.

Our study focuses on managing data and their replication taking all these parameters into account to improve the makespan 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. Our study is done in a steady state context. So we do not have to take into consideration initialisation costs, i.e. cost for transfering databases onto the server defined by the placement. Once the data is stored where it should be, it will not have to be moved again during the execution of jobs. From the data point of view, there are two possibilities for the platform. Either there is enough space on servers to store at least one copy of each database in the platform, or it is not possible and only a subset of databases can be stored at the same time.

We have computed a set of constraints describing the platform and its behavior. This leads to a linear formulation of the data placement and scheduling problem. The solution of this linear program gives us a placement for the databases on the servers as well as providing, for each kind of job, the server on which they should be executed. More precisely, for a type of request, we know how many jobs 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.

We also proved the NP-completeness of the problem (in the weak sense) by reduction to 2-PARTITION [23]. To find the optimal solution of the linear program, which is mixed with both rational and integer, we choose to solve the relaxed program over rational number. Then we designed an algorithm based on all parameters of the system to iteratively approximate the integer solution. Simulations (using OptorSim) proved that this algorithm is more efficient than a Greedy algorithm for the data mapping and an online algorithm for the scheduling of requests, especially when the storage space is small.

IV. Semi-Static Algorithm

Our first algorithm was based on the assumption that the frequency of requests and the load of the servers were known in advance (or if we were able to use the past to predict the future) and they did not change over time. This assumption holds for early experiments and local grids but this clearly not the case for large scale platforms and actual executions of real applications on large scale grids. Moreover, some specific accesses to databases by bioinformatic researchers can lead to large increases of accesses to some specific databases.

One first (and simple) solution could be to run our static algorithm once a while and redistribute the replicas according to the new replication scheme. The period has to be carefully computed because a small period will lead to an increase of the number of redistributions and a long period will lead to a bad load-balance of the platform. Moreover the computation the optimal mapping can be time-consuming. Another solution, is

to run the static algorithm once, and observe the platform to detect changes in data and computation task usages and to update the data mapping and request scheduling accordingly. The static algorithm can still be used for over larger periods of time.

The architecture of our replication and scheduling framework is given in Figure 1. Usual architectures of grid systems are made up of a resource broker, data manager, and computational servers. Resource manager receives computation requests from clients and distributes them to computational servers following a scheduling policy. Data manager is in charge of moving, deleting and locating data on the different servers. With SRA we have added a SRA manager that computes the static data mapping and scheduling using given information about the execution of previous requests. We now add a fifth component called DynSRA manager that observe load of computational servers and take dynamic decisions about placement and scheduling. These decisions are then sent to the SRA manager which gives order to resource broker and data manager.

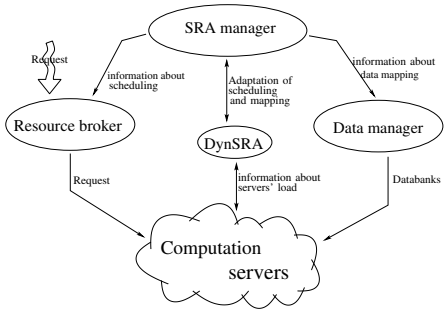

Fig. 1. Functional architecture of the DynSRA systems.

The semi-static algorithm is given in Algorithm 1. The idea is to found the server which the most loaded and balanced its load to other computation servers by replicating data and adapt scheduling. This algorithm can be sumarized in 5 main steps. For each step we have defined different metrics to determine the best server and data to consider in order to adapt scheduling and data placement.

a) Step 1 (Line 4): choice of overloaded server

First an overloaded server has to be chosen. We have several choices as a performance measure.

- **MaxCompServ**: most loaded server in computation volume. We compute the overall computation volume of requests to be solved on each server.
- **MaxTimeServ**: most loaded server in computation time. We take into account the server speed (we can use historical informations in case of external load).
- **MaxOverMedianServ**: server that is the most over from median in computation. We compare the load of a specific server compared to the average load over all the servers used in the target platform.

Then, we must choose the data which is the cause of overload.

Algorithm 1 General semi-static algorithm.

```
1:  while requests remain in the request queue do
2:    Compute the servers' load
3:    if there are overloaded servers then
4:      Choose P_overload as the most overloaded server
5:      Find the data D_load that leads to overload
6:      Choose P_underload an underloaded server
7:      if there is enough room to replicate D_load on
        P_underload then
8:        Replicate
9:      else
10:       while there is not enough room on P_underload for
          D_load do
11:         Choose a data to be removed on P_underload
12:       end while
13:       if there is still not enough room then
14:         Restore deleted files
15:         Choose another server as P_underload and go to
            step 1
16:       end if
17:     end if
18:     if if the data mapping has been changed then
19:       Update scheduling
20:       Resubmit requests concerned
21:     end if
22:   end if
23:   Wait for delay
24: end while
```

b) Step 2 (line 5): choice of data to be replicated to another server

- **MaxCompData**: most costly data in computation. We choose the data which is used by heavy load requests.
- **MaxDiffData**: data whose usage does not match expecting usage. We choose the data for which accesses are greater than expected during the computation of the static algorithm.

We need to find a server which can host the data to be replicated.

c) Step 3 (line 6): choice of underloaded server

- **MinCompServ**: less loaded server in computation. Same performance measure as in **MaxCompServ** but we choose the least loaded server.
- **MinTimeServ**: less loaded server in time. Same performance measure as in **MaxTimeServ** and we choose the least loaded server.
- **MaxUnderMedianServ**: server that that is the most over from median in computation. Same performance measure as in **MaxOverMedianServ**

If there is not enough room to put the data on a chosen server, one data must be deleted.

d) Step 4 (line 11): choice of data to delete if necessary

- **LessUsedData**: less used data.

- **MinCompData**: data that involves the less computation on chosen server.

Finally, one must update the scheduling of requests.

e) Step 5 (line 19): rescheduling

- **LPSched**: linear program with new placement fixed.
- **EqShareSched**: equal share of load between underloaded and overloaded servers.
- **MedianCompSched**: share between overloaded and underloaded servers depending of their current load.

All these choices for each step leads to different heuristics that may have different performance results. The different combinations and their results are discussed in the following section.

V. Performance Evaluation

To test the results of our model, we used Advanced OptorSim [24] which is simulator for grid environment dedicated to data management.

A. Experimental Environment

The target platforms of our studies are highly distributed and may span over multiple administrative domains. Therefore, it may be quite difficult to conduct repeatable experiments for long running applications on such systems.

For our experiments, we extracted from execution logs of NPS@ [25] cluster 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 used in SRA scheduling. This linear program is solved using *lp_solve*. The results give us information about data mapping and job scheduling. These outputs are used, with other configuration files, as inputs for the simulator.

The simulated platforms have been generated using Tiers[26]. Tiers has been widely accepted as a generator of realistic wide-area topologies. The topologies generated by Tiers have three level of hierarchy: WAN, MAN and LAN. We refer to [26] for a precise description of the topology generation procedure in Tiers. For the simulations, nodes at the LAN level are considered as computational clusters, nodes in MAN and WAN are only routers without computation or storage abilities.

Requests are submitted to the RB with a frequency around ten per second. This could seems to be a very high rate, but discussions with the biology and bioinformatic communities lead to the conclusion that the more computation power we provide, the more they will use.

B. Results and Discussion

In this section we discuss our experiments using Advanced OptorSim. We have done simulations for 14 combinations of heuristics. These combinations are described in Table I. In the following, we will refer to these combinations of heuristics using the id defined in Table I.

Our semi-static algorithm is designed to adapt scheduling and data placement when requests that are submitted do not match the usage pattern used for initial placement. So we have created a method that generates a set of requests that differ from another of a fixed value. Let v be that value, called the variation factor. A request is defined as an algorithm $a_k, k \in [1..p]$ applied on a databank $d_j, j \in [1..n]$. Starting from a known set of requests, we count $nb_{old}(j,k)$ the number of requests asking to apply a_k on d_j. Then we compute a new value $nb_{new}(j,k)$ so that:

$$nb_{new}(j,k) = nb_{old}(+/-)nb_{old} \cdot \frac{v}{100} \quad (1)$$

We now can calculate the new frequency of requests a_k on d_j that should appear on the new set.

$$f(j,k) = \frac{nb_{new}(j,k)}{\sum_i \sum_k nb_{new}(j,k)} \quad (2)$$

We then created 3 kinds of sets. In the first one, called all, the frequency of each kind of request has been modified and the fact that the new frequency was higher or lewer than initial was choosen randomly. In the second set, $max10$, only the frequency of the 10 kinds of request that were already the most used have been raised. In the third sets, $min10$, the frequency of the 10 less used kinds of request was increased.

1) Function of the Variation Factor v: Figures 2, 3 and 4 show the execution time of a set of 40000 requests as

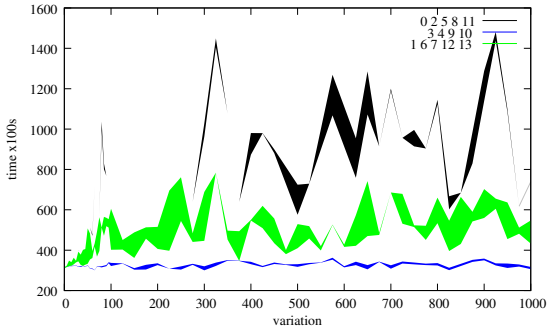

Fig. 2. Execution time for 40000 requests as a function of the difference between usage pattern use for initial mapping and scheduling and usage pattern that really occured for heuristics given in Table I

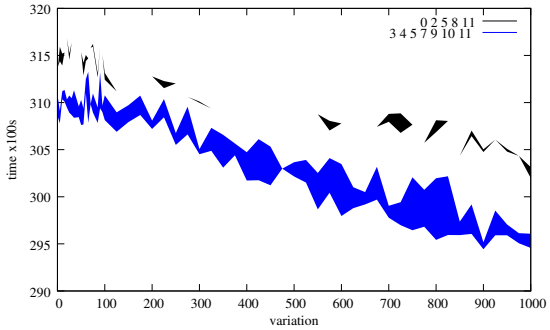

Fig. 3. Execution time for 40000 requests as a function of the difference between usage pattern use for initial mapping and scheduling and usage pattern that really occured for heuristics given in Table I. Only usage of 10 most frequent requests have been modified.

TABLE I

DIFFERENT COMBINATIONS THAT HAVE BEEN IMPLEMENTED IN THE SIMULATOR.

id	overload server	mostloaded data	underload server	data to delete	scheduling	id
0	none	none	none	none	none	0
1	MaxTimeServ	MaxCompData	MinTimeServ	none	EqShare	1
2	MaxTimeServ	MaxCompData	MinTimeServ	MinCompData	LPSched	2
3	MaxTimeServ	MaxCompData	MinTimeServ	MinCompData	EqShareSched	3
4	MaxTimeServ	MaxCompData	MinTimeServ	MinCompData	MedianCompSched	4
5	MaxTimeServ	MaxDiff	MinTimeServ	LessUsed	LPSched	5
6	MaxTimeServ	MaxDiff	MinTimeServ	LessUsed	EqShare	6
7	MaxTimeServ	MaxDiff	MinTimeServ	LessUsed	MedianCompSched	7
8	MaxOverMedianServ	MaxCompData	MinOverMedianServ	MinCompData	LPSched	8
9	MaxOverMedianServ	MaxCompData	MinOverMedianServ	MinCompData	EqShare	9
10	MaxOverMedianServ	MaxCompData	MinOverMedianServ	MinCompData	MedianCompSched	10
11	MaxOverMedianServ	MaxDiff	MinOverMedianServ	LessUsed	LPSched	11
12	MaxOverMedianServ	MaxDiff	MinOverMedianServ	LessUsed	EqShare	12
13	MaxOverMedianServ	MaxDiff	MinOverMedianServ	LessUsed	MedianCompSched	13

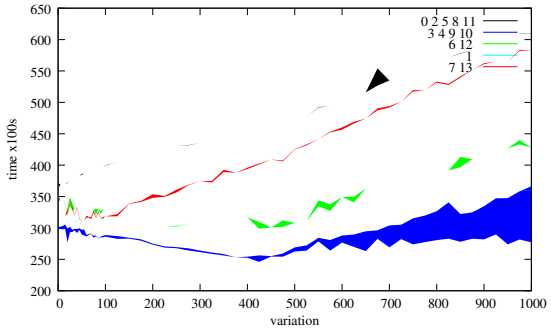

Fig. 4. Execution time for 40000 request as a function of the difference between usage pattern use for initial mapping and scheduling and usage pattern that really occured for heuristics given in Table I. Only usage of 10 less frequent requests have been modified.

a function of the variation factor used to generate the set of requests. As several heuristics have the same behaviour, we have joined them in the same shape to prevent to have 14 curves on the same graphics and make it more readable. Figure 2 is results for set of type all, Figure 3 for set of type $max10$, and Figure 4 for set of type $min10$.

On Figure 2, we can distinguish 3 distinct groups. Heuristics 3, 4, 9 and 10 let the system to have a stable execution time even when variation factor is very high. For the two other groups, execution time increases slowly when v between 1 and 100. After this point, results become really unstable and can be 5 times bigger than results of the best group.

In the group with worst results, there is the heuristic 0 which does not do any adaptation, and proves, if necessary the need of a dynamic mechanism. In the same group, there are all heuristics using **LPSched** . It can be explained by the computation time needed to solve the linear program but also, and mainly, by the fact that the system never have the good information about frequency usage. It causes the linear program to be not accurate to real situation and always make a wrong scheduling.

On Figure 3, we notice only two groups. It also shown that execution time is decreasing while the variation factor is increasing. That surprising result can be easily explained by the characteristics of requests affected by the variation factor. The most used requests are requests that need small computation power. So that, increasing the number of those small jobs will also reduce the number of biggest requests, and the computation power needed for the overall set of requests is lower. The group with worst performance is again composed of heuristics 0 and those using **LPSched** . All other heuristics have same results.

On Figure 4 there are five different groups. Once again, worst results are obtained by heuristic 0 and those using **LPSched** , for same previous reasons. The best results are obtained by heuristics using **MaxCompData** and **MinCompData** as for the all set. In that case, for variation factor between 1 and 400, heuristics 1, 3, 4, 6, 9, 10 and 12 have their execution time that decrease. As previously, it can be explained by the characterisitcs of requests affected by variation factor. The requests with the smallest frequencies are those that use biggest databanks. When one of these databanks is placed on a server, that one is almost dedicated to the few requests that need it (because there is not enough room to store other databanks). And the amount of computation power needed for those rare requests is lower than computation available on this server. Increasing the number of these kinds of requests will lead to a better use of these servers if the scheduling is adapted accordingly. But, when the frequency of these requests becomes too large to be executed on a single server, the biggest databanks should be replicated on other servers. Then the execution time for the whole set of requests increases, as we can seen in Figure 4 for variation factor higher than 400.

2) Function of Storage Space: Figures 5, 6, and 7 show the execution time of a set of 40000 requests as a function of the ratio between space available in the whole platform and the size of databanks to be stored. Like in previous figures, we have grouped heuristics which have the same behaviour in a single shape. Figure 5 is results for set of type all, Figure 6 for set of type $max10$, and Figure 7 for set of type $min10$.

In all three cases, we find the same results as in previous

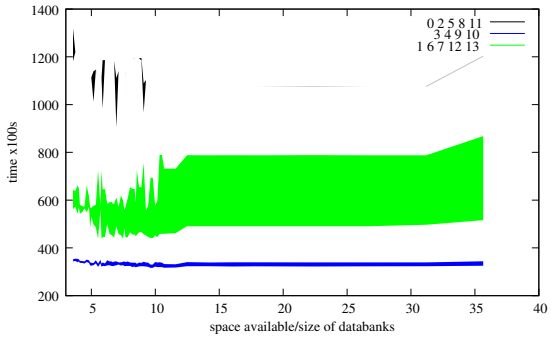

Fig. 5. Execution time for 40000 requests as a function of the ratio between space available in the whole platform and size of the databanks for heuristics given in Table I

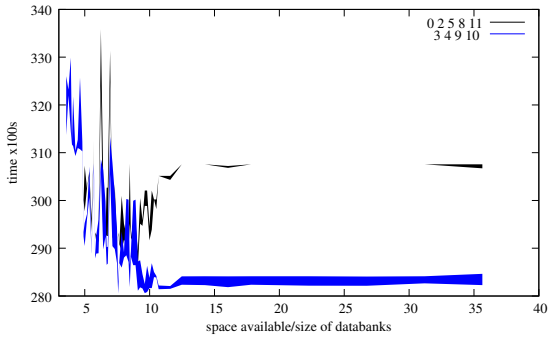

Fig. 6. Execution time for 40000 requests as a function of the ratio between space available in the whole platform and size of the databanks for heuristics given in Table I. Only usage of 10 most frequent requests have been modified.

section when storage space available is greater than 13 times the size of databanks to be stored in the platform. In that area, heuristics 0, 2, 5, 8 and 11 have always the worst results, while heuristics 3, 4, 9 and 10 have almost always the best results. It is to notice that heuristic 1 have better results than 3 and 9. The only difference between 1 and 3 is that 1 does not make any suppression of data. Added to the results of 4 and 9, we can conclude that the way scheduling is done is the

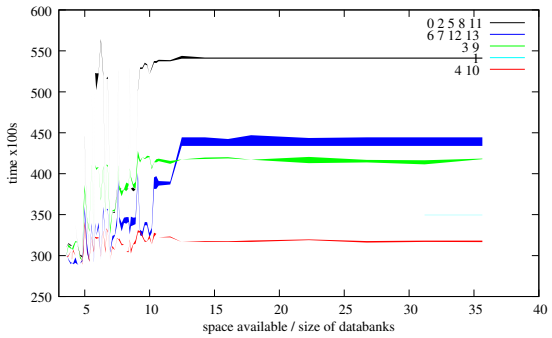

Fig. 7. Execution time for 40000 requests as a function of the ratio between space available in the whole platform and size of the databanks for heuristics given in Table I. Only usage of 10 less frequent requests have been modified.

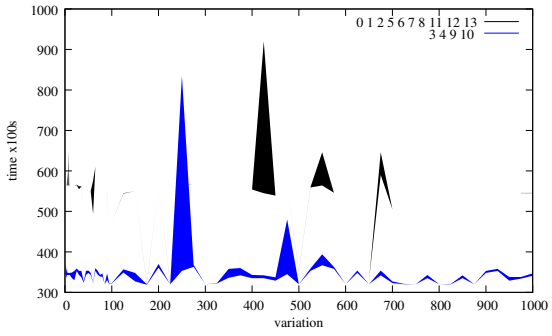

Fig. 8. Execution time for 40000 requests as a function of the difference between usage pattern use for initial mapping and scheduling and usage pattern that really occured for heuristics given in Table I. Initial data mapping is random.

most important but a bad choice of data to delete can results in too much data movement.

When the ratio between storage space available and the size of the databanks is in the interval $[1..12]$, results are very unstable. In the case of the all set, there are three distinct groups. In other groups, the difference between heuristics is not so important. With $max10$ and $min10$ sets, the same heuristics can have very different results. As it as already been seen in the case of a static placement [8], when storage space is very small, a data stored on a wrong server can have a big impact on performance of the overall platform.

3) Impact of the Initial Scheduling and Data Distribution: In order to see the impact of the initial distribution we ran the same simulation as in Section V-B.1. The difference is that, at start of the platform, databanks are randomly distributed amongst servers trying to fill all the space available in a greedy fashion. Each databanks is stored at least once and cannot be stored twice on the same server. All schedule informations are set to the same value (10). Figures 8 show the execution time of a set of 40000 requests for all set of requests. Like in previous figures, we have grouped heuristics which have the same behaviour in a single shape.

In Figure 8, we can notice two groups of heuristics. Once again, best results are obtained for heuristics 3, 4, 9 and 10. It means heuristics using **MaxCompData** and **MinCompData** for the choice of data to replicate and delete and **EquShareSched** or **MedianCompSched** for rescheduling. With random distibution, performance are lower than starting with SRA placement and scheduling. The main conclusion of these simulations is that the heuristics 3, 4, 9, and 10 allow the system to keep the execution time quite stable whatever the initial distribution is.

We have done same experiments with $min10$ and $max10$ sets of requests (not shown here). In both case the two same groups of heuristics with same kind of results as with all set can be observed.

C. Summary of Experimental Results

In the three series of experiments, we have evaluated the impact of the different heuristics designed to detect a bad

load-balancing among the available servers and adapt the data placement and request scheduling.

The random series prove that even when variation factor is very high, the initial placement has an importance. Starting the simulation with a random distribution does not, in most case, give result as good as starting with SRA placement and scheduling.

In all case, the heuristics based on **MaxCompData** , **MinCompData** and **MedianCompSched** gave the best results allowing to balance load between the different servers. Heuristics using **LPSched** always give the worst results. The linear program has been designed for a static use, and it never computes good scheduling information when it has a knowledge about pattern usage of databanks which is not accurate enough.

But when the storage available in the whole platform is quite small, it is quite difficult to adapt correctly data distribution and scheduling if the main load is caused by small ($max10$) or very big ($min10$) databanks.

VI. Conclusion and Future Work

In this paper, we have presented several ways of optimizing a static data mapping and request scheduling algorithm when the requests frequency and the load of the overall platform changes over time. Following some recent results on a join replication and scheduling algorithms, we have proposed a new semi-static algorithm that allows an optimization of the data mapping and request scheduling at run-time.

Using Advanced Optorsim simulator, we have tested a set of heuristics designed to adapt scheduling and data placement to change in data usage patterns. Heuristics are based in observation of historical performance data on computation servers to detect overloaded servers and how redistribute their load to other servers in order to optimize the use of computational power available in the platform. Simulations shown that the good choice of data that cause the overloading of server and replicate it on the right server let the system to stabilize execution time of the whole set of requests with the condition to adapt correctly scheduling information. Simulations using **MaxCompData** , **MinCompData** and **MedianCompSched** had always the best performance results.

Our future work will consist of adding communication costs for the requests in the model of the SRA and design new heuristics to take them into account. We also plan to implement these heuristics in the DIET [27] environment to validate them in a real environment on the Grid'5000 platform.

Acknowledgements

This work was supported in part by the ACI GRID and Grid'5000 projects of the French Department of Research and the European project Coregrid.

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.

References

[1] F. Berman, G. Fox, and A. Hey, Eds., *Grid Computing: Making the Global Infrastructure a Reality*. Wiley, 2003.

[2] I. Foster and C. Kesselman, Eds., *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2004.

[3] 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*, vol. 23, pp. 187–200, 2001.

[4] X. Qin and H. Jiang, "Data Grid: Supporting Data-Intensive Applications in Wide-Area Networks," University of Nebraska-Lincoln, Lincoln, NE, USA, Tech. Rep. TR-03-05-01, May 2003.

[5] 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*, Dec. 2000.

[6] A. Krishnan, "A Survey of Life Sciences Applications on the Grid," *New Generation Computing*, vol. 22, pp. 111–126, 2004.

[7] F. Desprez and A. Vernois, "Simultaneous Scheduling of Replication and Computation for Bioinformatic Applications on the Grid," in *CLADE 2005*. IEEE Computer Society Press, July 2005.

[8] ——, "Simultaneous Scheduling of Replication and Computation for Data-Intensive Applications on the Grid," *Journal Of Grid Computing*, vol. 4, no. 1, pp. 19–31, Mar. 2006.

[9] W. Bell, D. Cameron, L. Capozza, A. Millar, K. Stockinger, and F. Zini, "OptorSim - A Grid Simulator for Studying Dynamic Data Replication Strategies," *International Journal of High Performance Computing Applications*, vol. 17, no. 4, 2003.

[10] C. Xu, H. Jin, and P. Srimani, "Special Issue on Scalable Web Services and Architecture," *J. of Parallel and Dist. Comput.*, vol. 63, 2003.

[11] V. Cardellini, E. Casalicchio, M. Colajanni, and P. Su, "The State of the Art in Locally Distributed Web-Server Systems," *ACM Computing Surveys*, vol. 34, no. 2, pp. 263–311, June 2002.

[12] S. Podlipding and L. Bszrmenyi, "A Survey of Web Cache Replacement Strategies," *ACM Computing Surveys*, vol. 35, no. 4, pp. 374–398, Dec. 2003.

[13] H. Lamehamedi, B. Szymanski, Z. Shentu, and E. Deelman, "Data Replication Strategies in Grid Environments," in *Proc. 5th Int. Conf. on Algorithms and Architecture for Parallel Processing, ICA3PP'2002*. IEEE Computer Science Press, Oct. 2002, pp. 378–383.

[14] "The European DataGrid Project," http://www.eu-datagrid.org.

[15] R. C.-S. D.G. Cameron, A. Millar, C. Nicholson, K. Stockinger, and F. Zini, "Evaluating Scheduling and Replica Optimisation Strategies in OptorSim," in *4th International Workshop on Grid Computing (Grid2003)*. IEEE Computer Society Press, Nov. 2003.

[16] W. Bell, D. Cameron, L. Capozza, A. 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)*, ser. Lecture Notes in Computer Science. Springer Verlag, Nov. 2002.

[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, Mar. 2004.

[18] D. Thain, T. Tannenbaum, and M. Livny, "Distributed computing in practice: The condor experience," *Concurrency and Computation: Practice and Experience*, 2004.

[19] H. Mohamed and D. Epema, "An Evaluation of the Close-to-Files Processor and Data Co-Allocation Policy in Multiclusters," in *Cluster 2004*. IEEE Computer Society Press, 2004, pp. 287–298.

[20] 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.

[21] ——, "Simulation Studies of Computation and Data Scheduling Algorithms for Data Grids," *Journal of Grid Computing*, vol. 1, no. 1, pp. 53–62, 2003.

[22] A. Chakrabarti, R. Dheepak, and S. Sengupta, "Integration of Scheduling and Replication in Data Grids," Infosys Tech. Ltd, Tech. Rep. TR-0407-001, July 2004.

[23] M. R. Garey and D. S. Johnson, *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[24] "Advanced Optorsim," http://lipforge.ens-lyon.fr/projects/advancedoptor/.

[25] C. Combet, C. Blanchet, C. Gourgeon, and G. Delage, "Nps@: Network protein sequence analysis," *TIBS*, vol. 25, No 3, pp. [291]:147–150, Mar. 2000.

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

[27] E. Caron and F. Desprez, "DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid," *International Journal of High Performance Computing Applications*, vol. 20, no. 3, pp. 335–352, 2006.