Simultaneous Scheduling of Replication and Computation for Bioinformatic Applications on the Grid[⋆]

Frédéric Desprez[1], Antoine Vernois[1], and Christophe Blanchet[2]

[1] LIP, UMR CNRS-INRIA-UCBL 5668
ENS Lyon, France
{Frederic.Desprez,Antoine.Vernois}@ens-lyon.fr
[2] LBRS Laboratory / IBCP Institute
UMR 5086 CNRS, Univ. Claude Bernard Lyon 1
7 passage du Vercors, 69007 Lyon, France
Christophe.Blanchet@ibcp.fr

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). The PattInProt application searches sites and signatures of proteins into databanks of protein sequences.

1 Introduction

One of the first motivations of using grids [11, 21] comes from applications managing large data sets [17, 31] such in Life Science [7, 24, 26] or for example in High Energy Physic [23]. Indeed, life Science is a scientific field that produce continuously lot of data through experiences such as complete genome sequencing projects (1220 projects in november 2004 [12]). These raw bioinformatic datasets come generally from different sources located in different institutes, and need to be analyzed with many different algorithms [29]. Grid is a good mean to solve the equation of analysing such large datasets with a large panel of bioinformatic software. 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

[⋆] This work was supported in part by the ACI GRID of the french department of research

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 (for example pattern scanning, searching for signature or functional site of 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 experimentation using the OptorSim simulator [9] for replica managers.

2 Motivating Example

Our motivation for this work comes from the PattInProt application about the search of sites and signatures of proteins into databanks of protein sequences.

Genomic acquiring programs such as full genomes sequencing projects produce large amounts of data, made available to the community. These raw data have to be understood and annotated in order to be useful for further studies and for cross references to and from other datasets. There is also a large number of bioinformatic tools used to analyze these data, and they come from different fields of Bioinformatic (similiraty and homology, protein function analysis, sequence analysis, etc). But many of them can be modeled as shown in Figure 1. Protein function analysis, such as the PattInProt application studied in a grid context by the GriPPS project, can act as a good model of such a bioinformatic application requiring access to several datasets of various sources and sizes.

Functional sites and signatures of protein are very useful for analyzing these data or for correlating different kinds of existing biological data. Sites and signatures of protein can be expressed using the syntax defined by the PROSITE [14] databank, and written as a 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 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 be very high as the number of users increases every day thanks to the Internet.

The difficulties come from the fact that the number of datasets used as reference for this kind of search can be large and of very different sizes. These datasets can be international protein sequence databanks such as Swiss-Prot/TrEMBL [13], PIR [34], etc. But they also can be raised quite directly from genome sequencing projects with the translation of the CDS (CoDing Sequence) extracted from the gene sequence obtained. In this case, the number of datasets are as large as the genome project [12], and as variable as the size of the genomes (e.g. 3 Gpb for the human genome or 120 Mpb for the Saccharomyces cerevisiae genome).

Figure 1 describes the classical architecture of a bioinformatic application. We can notice two kinds 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 such as EBI [1], NCBI [3], SIB [5], or NPS@ [4]. Clients access computational servers through web portals or directly by asking for an account to servers administrators.

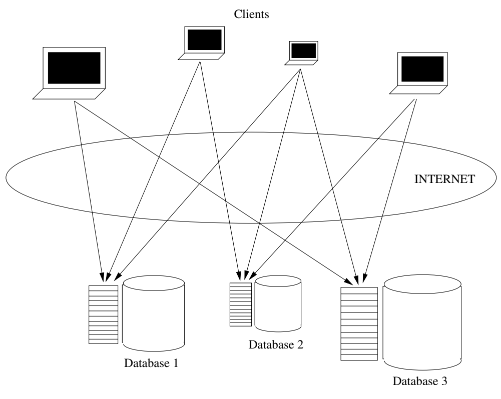

Fig. 1. Current view of a Bioinformatic Application

We accessed the logs of such a cluster that provides computational work through a bioinformatic web portal and allows users to apply some well known algorithms to existing databanks. The portal is the “Network Protein Sequence Analysis - NPS@” bioinformatic portal [18], providing around 40 algorithms and 7 databanks to biologists and bioinformaticians for biological queries. NPS@ is up since 1998 and had answered to more than 6 million bioinformatic analyses. It currently computes more than 3 thousands of such analyses per day. The portal and the cluster are located at IBCP [2] in Lyon (France), a research institute on biology and chemistry of proteins, and they are managed by the bioinformatic team of this laboratory.

Input of such requests are user’s protein sequences or signatures that usually do not excess a few kilobytes. This is a centralized cluster with limited capacities so only major databanks and algorithms are available. This is currently improved by the GriPPS [22] which aims at distributing its work among a large number of servers made accessible through the grid.

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 [35]. Several researches have studied how to optimally replace data in distributed web caches [15, 30]. 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 [28] and among them the researches for the Datagrid project from the CERN [6]. OptorSim [9, 19] allows to simulate data replication algorithms over a grid. This tool is more precisely described in Section 5.1. In [8], 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. OGSA [27] also proposes a replication service which is currently not connected to request scheduling. In [25], 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.

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 [32, 33], 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 [16], 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).

4.1 Model

Our model is based on three kinds of objects: a set of computational servers P_i, $i \in [1..m]$, a set of data d_j of size $size_j$, $j \in [1..n]$ and 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 it is possible to store at least one replica of each data.

Our goal is to find a placement of the databanks that maximizes the throughput of the platform. We call TP this throughput. It 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.

The throughput is limited by some constraints due to the specifications of the platform and the requests. First, the space on each server is limited by its storage capacity. So the total size of data stored on server P_i cannot exceed m_i Then 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. 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.

Let $\delta_i^j = 1$ if there is a replica of data d_j on server P_i, $\delta_i^j = 0$ otherwise. Considering previous constraints, we can define the linear program of Figure 2. 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

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

Fig. 2. Linear Program Formulation

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 A Greedy Solution

Starting with the same platform and algorithm models, we also design a greedy algorithm 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 data are sort 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.

5 Experiments

To experiment the results of our model, we used OptorSim [9, 19], a simulator of Data Grid environments developed in the Work Package 2 of EU Datagrid project [6]. We have modified OptorSim to exactly match our needs.

5.1 Experimental Environment

The target platforms of our studies are distributed and may span multiple administrative domains. Therefore, it may be quite difficult to conduct repeatable experiments for long running applications on such systems. So we choose to make use of simulation to experiments our algorithms. Other advantage to use simulation is the ability to easily test our algorithm with different network configurations which is not possible using experiments on a real platform.

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 Mflops. All Worker Nodes managed by the same CE have the same capacity of computation, but WN from different CE may have different capacities.

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). The same file can be stored on different SEs at the same time. 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) is in charge of all data movements between sites. And finally, jobs are created and scheduled by the Resource Broker which is able to instantiate communications with CE and RM to get all information needed about SE, network bandwidth, job queues, etc. for scheduling purpose. In our case, a job is defined by an algorithm and a databank on which the algorithm is applied.

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* [10]. 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 inspired from the architecture of the European DataGrid testbed. There are ten clusters of eight nodes with associated SE and seven routers without any storage nor computation 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 biologist and bioinformatic community lead to the conclusion that the more computation power we can give them, the more they will use.

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* (for Minimum Completion Time) 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.

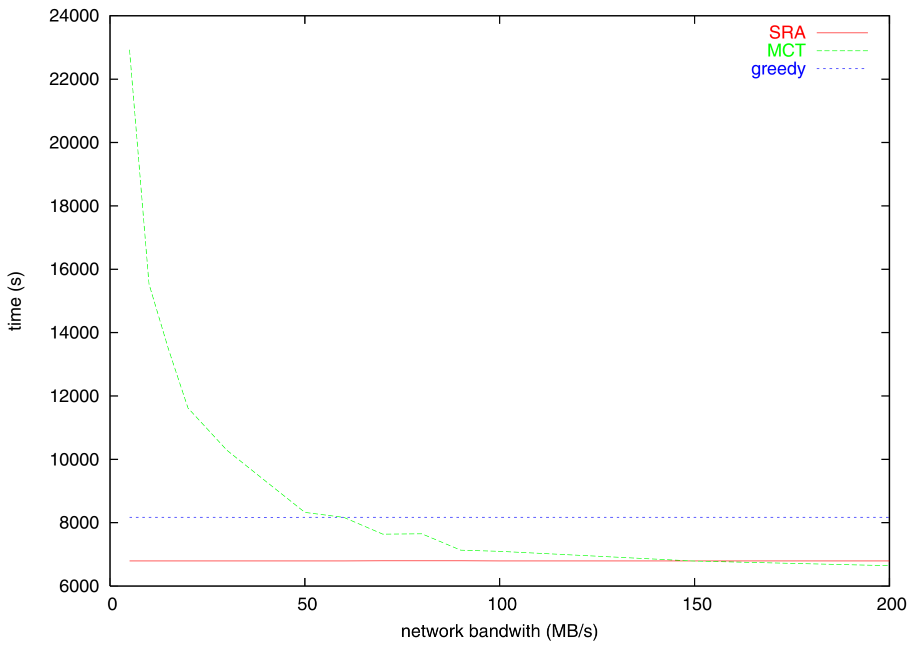

Fig. 3. Execution time for 40000 jobs as function of the network bandwidth

Figure 3 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. On Figure 3, 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.

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, the scheduling uses an on-line MCT

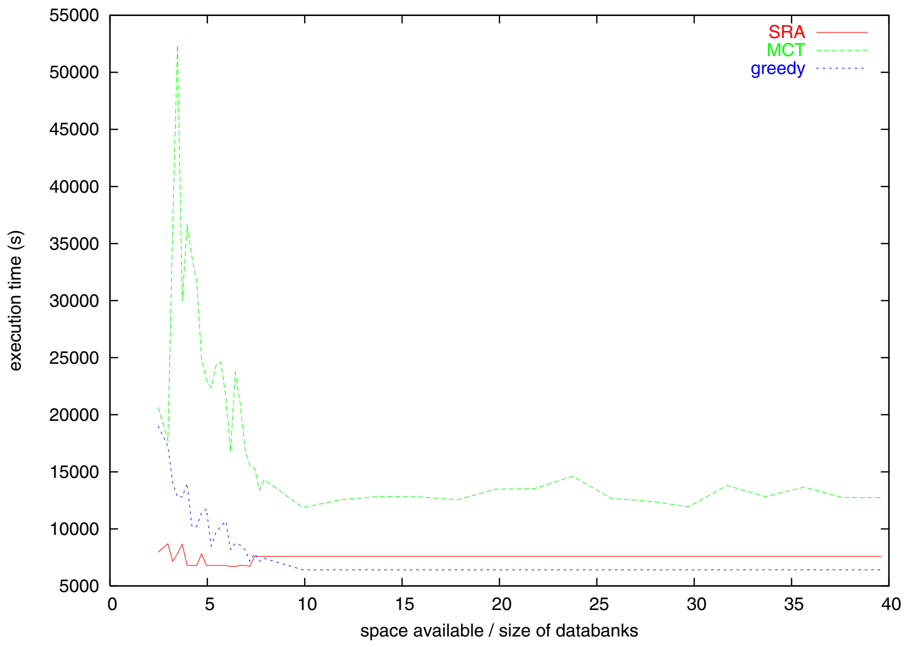

Fig. 4. Execution time for 40000 jobs as a function of available space on SE

method but there is still no data movement because the algorithm totally fills the available space in the platform. So the scheduler always schedules requests where a data is available. As MCT favours the execution time of current request to schedule, it does a lot of transfers. 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 4 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.

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.

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 work 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 [20] environment to deploy efficiently the GriPPS [22] application. A replica manager will be designed and developed in this environment.

Acknowledgements

The authors would like to thanks A. Legrand, L. Marchal and Y. Robert for their work on steady-state scheduling and discussion about the model used in this article. They also would like to thank C. Combet for his work and the access on the NPS@ bioinformatic portal.

References

1. EBI. http://www.ebi.ac.uk.
2. Inst. de Biologie et Chime des Protéines. http://www.ibcp.fr.
3. NCBI. http://www.ncbi.nlm.nih.gov.
4. NPS@. http://npsa-pbil.ibcp.fr.
5. SIB. http://www.isb-sib.ch/.

6. The European DataGrid Project. http://www.eu-datagrid.org.
7. R. Apweiler, A. Bairoch, and C. H. Wu. Protein sequence databases. *Current Opinion in Chem. Bio.*, 8:76–80, 2004.
8. 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. Workshop on Grid Comput. (Grid'2002)*. Springer Verlag, Nov. 2002.
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*, 17(4), 2003.
10. M. Berkelaar. LP_SOLVE. http://www.cs.sunysb.edu/~algorith/implement/lpsolve/implement.shtml.
11. F. Berman, G. Fox, and A. Hey, editors. *Grid Computing: Making the Global Infrastructure a Reality*. Wiley, 2003.
12. A. Bernal, U. Ear, and N. Kyrpides. Genomes OnLine Database (GOLD): A Monitor of Genome Projects World-Wide. *NAR*, 29:126–127, 2001.
13. B. Boeckmann, A. Bairoch, R. Apweiler, M.-C. Blatter, A. Estreicher, E. Gasteiger, M. Martin, K. Michoud, C. O'Donovan, I. Phan, S. Pilbout, and M. Schneider. The SWISS-PROT Protein Knowledgebase and its Supplement TrEMBL in 2003. *Nucleic Acids Res.*, 31:365–370, 2003.
14. P. Bucher and A. Bairoch. A Generalized Profile Syntax for Biomolecular Sequences Motifs and Its Function in Automatic Sequence Interpretation. In *Proceedings 2nd International Conference on Intelligent Systems for Molecular Biology*, volume 2, pages 53–61. AAAIPress, 1994.
15. V. Cardellini, E. Casalicchio, M. Colajanni, and P. Su. The State of the Art in Locally Distributed Web-Server Systems. *ACM Computing Surveys*, 34(2):263–311, June 2002.
16. A. Chakrabarti, R. Dheepak, and S. Sengupta. Integration of Scheduling and Replication in Data Grids. In *Proceedings 11th International Conference on High Performance Computing (HiPC 2004)*, pages 375–385. Springer, Dec. 2004.
17. 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. *J. of Netw. and Comp. Appl.*, 23:187–200, 2001.
18. C. Combet, C. Blanchet, C. Gourgeon, and G. Deléage. Nps@: Network protein sequence analysis. *TIBS*, 25, No 3:[291]:147–150, Mar. 2000.
19. 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.
20. DIET. http://graal.ens-lyon.fr/DIET/.
21. I. Foster and C. Kesselman, editors. *The Grid 2: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2004.
22. GriPPS. http://gripps.ibcp.fr.
23. 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)*, Dec. 2000.
24. N. Jacq, C. Blanchet, C. Combet, E. Cornillot, L. Duret, K. Kurata, H. Nakamura, T. Sil-vestre, and V. Breton. Grid as a Bioinformatic Tool. *Parallel Comp.. Special issue: High-performance parallel bio-comp.*, 30(9-10):1093–1107, 2004.
25. 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)*, Mar. 2004.

26. A. Krishnan. A Survey of Life Sciences Applications on the Grid. *New Generation Computing*, 22:111–126, 2004.
27. P. Kunszt and L. Guy. *Grid Computing: Making the Global Infrastructure a Reality*, chapter The Open Grid Services Architecture for Data Grids, pages 385–435. Wiley, 2003.
28. H. Lamehamedi, B. Szymanski, Z. 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. IEEE Computer Science Press, Oct. 2002.
29. G. Perriere, C. Combet, S. Penel, C. Blanchet, J. Thioulouse, C. Geourjon, J. Grassot, C. Charavay, M. Gouy, L. Duret, and G. Deleage. Integrated Databanks Access and Sequence/Structure Analysis Services at the PBIL. *Nucleic Acids Res.*, 31:3393–3399, 2003.
30. S. Podlipding and L. Böszörmenyi. A Survey of Web Cache Replacement Strategies. *ACM Computing Surveys*, 35(4):374–398, Dec. 2003.
31. X. Qin and H. Jiang. Data Grid: Supporting Data-Intensive Applications in Wide-Area Networks. Technical Report TR-03-05-01, Univ. of Nebraska-Lincoln, May 2003.
32. K. Ranganathan and I. Foster. Decoupling Computation and Data Scheduling in Distributed Data Intensive Applications. In *Proc. of the 11th Int. Symp. for High Performance Distributed Computing (HPDC-11)*, July 2002.
33. 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.
34. C. Wu, L. Yeh, H. Huang, L. Arminski, J. Castro-Alvear, Y. Chen, Z. Hu, P. Kourtesis, R. Ledley, and B. e. a. Suzek. The Protein Information Resource. *Nucleic Acids Res.*, 31:345–347, 2003.
35. C. Xu, H. Jin, and P. Srimani. Special Issue on Scalable Web Services and Architecture. *Journal on Parallel and Distributed Computing*, 63, 2003.