Application Re-Strusturing and Data Management on a GRID Enviroment: a Case Study for Bioinformatics

Giovanni Ciriello[1], Matteo Comin[1], Concettina Guerra[1,2]

[1]University of Padova
Dept. of Information Engineering
Via Gradenigo, 6/B 35131 Padova
{ciriello, ciompin, guerra}@dei.unipd.it

[2]Georgia Institute of Technology
College of Computing
Atlanta, GA 30332-0280 USA
guerra@cc.gatech.edu

Abstract

This paper describes a distributed implementation of PROuST, a method for protein structure comparison, that involves a major restructuring of the application for an efficient grid immersion. PROuST consists of several components that perform different tasks at different stages. Given a target protein, an index-based search retrieves from a database a list of proteins that are good candidates for similarity, then a dynamic programming algorithm aligns the target protein with each candidate protein. The same geometric properties of secondary structure elements of proteins are used by different components of PROuST. Thus, an important issue of the distributed implementation is data transfer vs. data recomputation tradeoffs. Our implementation avoids recomputation by re-using the hash table data as much as possible, once they are accessed. The algorithmic changes to the application allow to reduce the number of data accesses to storage elements and consequently the execution time. In addition this paper discusses data replication strategies on a grid environment to optimize the data transfer time.

1. Introduction

Comparing protein structures is important for protein classification and for understanding the protein functions. We have developed a method PROuST [3] that allows efficient retrieval of similarity information from a database containing all protein structures of the Protein Data Bank PDB [1]. This paper presents a distributed implementation of PROuST that involves a major restructuring of the application for an efficient grid immersion. PROuST consists of two main components: the first component generates hypotheses of similarity for a target protein in an efficient way using an index-based search, while the second component verifies these hypotheses by performing pairwise comparisons and alignments by dynamic programming (DP). The index-based search uses an indexing technique that involves geometric properties of the secondary structure elements (SSE) of the proteins, i.e. α-helices and β-strands. It computes for all triplets of SSE the angles formed by the three pairs of linear segments associated to the SSE and uses them as indexes to a three-dimensional (3D) hash table. The hash table, built in a preprocessing step, stores all triplets of all proteins of the PDB. Once built, it allows to retrieve a list of proteins that are good candidates for similarity with the target protein, without the need to examine separately every single protein of the PDB. A pairwise structure comparison is then performed to align the target protein with each protein of the candidate list. By combining these two approaches, PROuST achieves good results in terms of robustness and agreement with existing classifications of protein structures. In addition its time performance compares well with that of other existing approaches. However, the amount of computation and data involved is quite high, given the current size of the PDB of more than 33,000 structures. For an exhaustive analysis of classification accuracy and time performance we refer the reader to [3].

Previous work [4] has included a distributed implementation of the index-based search. The new distributed implementation integrates the index-based search and the dynamic programming algorithm, and it requires some major changes to the algorithms and data structures resulting in a more efficient solution. It exploits the fact that the same geometric properties of secondary structure elements of proteins, angles and distances, are used by the two main components of PROuST. Obviously, an important issue of the distributed implementation is data transfer vs. data recomputation tradeoffs. The new solution avoids recomputation by re-using the hash table data as much as possible, once they are accessed.

The emerging grid technology [5] is becoming an unavoidable aspect for the solution of compute intensive problems. The computational grid enables the use of a large number of different machines acting as a single one, by sharing both storage capacity and computing power. The importance of sharing data and resources in a secure manner is proved by the increasing interest of scientist towards this technology, especially in the biomedical community, that includes bioinformatics and medical area [9]. The Grid.it project [2], within which this work has been done, is aimed at developing a middleware layer enabling science in areas such as High Energy Physics, Earth Observation and Biomedicine. Biomedical applications have very challenging requirements in terms of computational power and amount of data. This paper focuses on a structural bioinformatics application on a grid and deals with the issues of data management and algorithmic enhancement for an efficient porting of the application onto the grid. In addition, it discusses data replication strategies to optimize the data transfer time. Replica management is a crucial aspect of a gridifying strategy [7, 8], because the availability of data in Storage Elements (SE) close to the Computing Element (CE) where the job requesting the data runs enables latency reduction and efficient access.

The paper is organized as follows. In the next section we review the method PROuST. In section 3 we describe a restructuring of the application that reduces the number of data accesses and avoids recomputation. In section 4 the immersion of the application in a grid is discussed. The time performance is analyzed in section 5. This work ends with conclusions in section 6.

2. PROuST: an all-to-all protein structure comparison method

PROuST [3] combines different techniques that allow fast retrieval of similarity information from a database containing all the protein structures of the PDB. Proteins from the PDB are represented by linear segments associated to their SSE, i.e. α-helices and β-strands. In our internal representation, the proteins from PDB are encoded by a set of files, each consisting of the list of the SSE segments of a protein. Starting from these files, all triplets of SSE are computed, and the angles formed by the three pairs of segments of a triplet are used as indexes to a 3D hash table, where the triplets are stored. Once the hash table is built, an entry or cell contains the triplets of all proteins characterized by similar dihedral angles. The hash tables are accessed to retrieve very efficiently the proteins most similar to a given target protein. This is done as follows. For each triplet of segments of the target protein, its three angles are used to access the corresponding entry of the hash table; for each triplet belonging to the indexed cell and equivalent to the target protein (see below), a counter is incremented for the protein containing it. As a result, a list of candidate proteins ranked according to the values of these counters is generated.

The second component of PROuST is a pairwise protein structure comparison which aligns the target protein with each candidate protein of the list. It uses as input the internal representation of the proteins, and returns a similarity score according to which the candidate proteins are re-ranked into the final similarity list. The alignment is obtained by DP. If P is the target and Q a candidate protein, with SSE segments $p_1, \cdots, p_n$ and $q_1, \cdots, q_m$, respectively, DP finds the associations (p_i, q_j) that maximize a given similarity score and also satisfy the continuity constraints, i.e. if (p_i, q_j) and (p_h, q_k) are two pairs of associated SSE and $i < h$ then $j < k$. The DP determines the optimal non-decreasing path in a 2D score matrix M. The main characteristic of our algorithm, that distinguishes it from other DP algorithms applied to SSE alignment, is that the score matrix is built from geometric properties of triplets of secondary structures, the same properties used by the indexing procedure. The score of the pair (p_i, q_j), stored at $M(i, j)$, is given by the number of times the pair (p_i, q_j) occurs in any two equivalent triplets of SSE segments of P and Q. Two triplets (p_u, p_v, p_z) and (q_r, q_s, q_t) are *equivalent* if they have similar angle and distance values. Two equivalent triplets determine three candidate pairs of corresponding segments: (p_u, q_r), (p_v, q_s), and (p_z, q_t), that contribute to the entries $M(u, r)$, $M(v, s)$, $M(z, t)$ of score matrix M, whose values are incremented by one. As we will see in the next section, the construction of the score matrix M is the step most affected by the restructuring of PROuST.

All components of PROuST are shown in the workflow of Figure 1. The last component determines a superposition at the atomic level of the target protein with any protein selected from the similarity list.

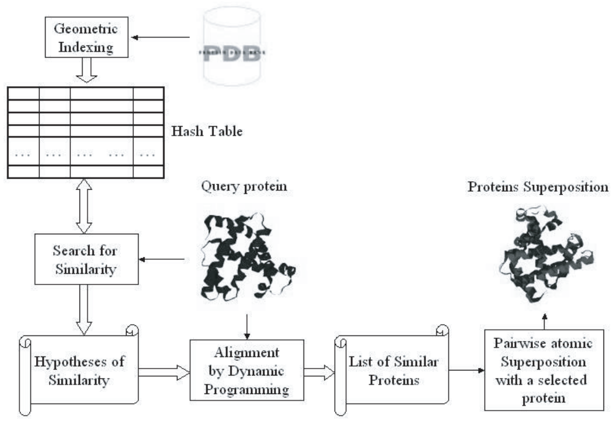

Figure 1. PROuST workflow: the tasks for structural comparison of proteins.

2.1. A distributed implementation

The construction of the hash tables is compute intensive, due to the large number of proteins in the PDB inserted in the hash table (currently, more than 33,000), with a storage requirement for the hash table of more than 5 Gbytes. A distributed implementation of the hash table construction is based on a partition of the hash table into subtables, each containing a subset of the proteins of the PDB. The implementation achieves a good load balance and minimizes the number of accesses to secondary storage; a detailed description of this step can be found in [6].

The index-based search for similarity for a target protein follows a master/slave paradigm. The search is carried out independently by all slaves each operating on a different subtable. Each slave generates a list of candidate proteins. The master collects from the slaves all such lists and then merges them to obtain the overall ranking. The final candidate list produced by the index-based search varies in length, but, typically, for a protein with an average number of SSE (i.e. 12-13 SSE) consists of more than 5000 proteins. The next DP component is performed on either the entire list of candidates or a reduced list of top-ranked candidates. The DP uses as input the internal vectorial representation of the proteins to generate the score matrices, based on which the similarity score is determined. Even though the DP computation for a pair of proteins is quite fast, and the list of proteins has been reduced from 33,000 thousands proteins of the PDB to a few thousands candidates, the amount of time required by all DP alignments may still be quite high. Obviously, all DP computations can be done in parallel on all candidates. An important issue arises however in the distributed implementation of DP, that is whether to exploit and reuse the information that has been already computed and stored in the hash table or recompute it from the internal representation of proteins. This issue will be discussed in detail in the next section.

We remark here that often only a set of representative proteins is selected from the PDB for structure comparison, in such cases obviously the time requirements go down considerably. We choose to search the entire PDB because we want to discover non-trivial similarities, not only related to the standard family classification.

3. Re-structuring PROuST

In the previous section we described PROuST and its main components:

1. hash table construction
2. index-based similarity search
3. pairwise structural alignment
4. atomic superimposition.

Here we describe a major restructuring of our application for an efficient grid immersion. As already mentioned, the same geometric properties of SSE, angles and distances, are used by different components of PROuST. Thus, we need to address the issue of data transfer vs. data recomputation tradeoffs. In our initial implementation, angles and distances of all triplets of SSE were recomputed by the structural alignment procedure for each pair of target and candidate proteins, even though they were available in the hash tables. It should be noted that the recomputation of the geometric properties for all candidate proteins also requires access to many small files each containing the vectorial representation (list of SSE segments) of a protein. In a local scenario, recomputation turned out to be more cost-effective, due to the large size of the hash table and large number of elements in each table cell. With more than 30,000 proteins, the number of triplets in the hash table is above 40,000,000. Consequently, the access time for the table is relatively large. Moving from a local to a distributed system data transfers became a crucial aspect. We propose a new solution that avoids recomputation by re-using the hash table data as much as possible, once they are accessed. It combines indexing and structural alignment: the index-based search in addition to generate a candidate list of proteins also builds the score matrices to be used later by the pairwise structural comparisons.

The following is a sketch of the unified procedure that, given the target protein P, returns the list $\mathcal{C}$ of candidate proteins and the list $\mathcal{L}$ of score matrices. We denote by M_Q the score matrix for the pair of proteins P and Q. M_Q is created only if protein Q has at least one triplet of SSE equivalent to a triplet of P.

```
UNIFIED PROCEDURE
Step 1.
Initialize the list $\mathcal{L}$ and $\mathcal{C}$ to empty.
Step 2.
Consider all triplets of SSE of $P$ and for each such triplet $(p_u, p_v, p_z)$, with $u < v < z$, do the following:
  i. Compute the angles $\alpha_{uv}, \alpha_{vz}, \alpha_{uz}$ and the distances $d_{uv}, d_{vz}, d_{uz}$ of the three pairs of segments.
     Access the hash table at the cell indexed by the three quantized angles.
  ii. If the cell is not empty, scan all triplets of SSE stored in the cell.
      Let $(q_r, q_s, q_t)$ be one such triplet, $r < s < t$, with distances $h_{rs}, h_{st}, h_{rt}$, and $Q$ the protein containing $q_r, q_s$, and $q_t$.
      if the distances of the two triplets are within a given threshold TD , i.e.
         $|d_{uv} - h_{rs}| < TD$, $|d_{vz} - h_{st}| < TD$ and $|d_{uz} - h_{rt}| < TD$ then
         ii.a Index-based Search
            Cast a vote to protein $Q$ and insert it into list $\mathcal{C}$ if not present.
         ii.b Score matrix building
            If $\mathcal{L}$ does not contain an entry for protein $Q$, create a new score matrix $M_Q$, initialized to 0, and insert it into $\mathcal{L}$.
            Update $M_Q$ as follows:
               $\{M_Q(u,r) = M_Q(u,r) + 1;\ M_Q(v,s) = M_Q(v,s) + 1;\ M_Q(z,t) = M_Q(z,t) + 1;\}$
End.
```

The details of the voting process are omitted here (see [6]). The above procedure builds all score matrices at the same time; with a protein of average size, this may imply the creation and update of more than 5000 data structures. To guarantee reasonable storage requirements, we use an ad-hoc dynamic data structure for storing a matrix, called *Dynamic Matrix* (DyM). A score matrix is represented as a linked list of column vectors each of length n (n being the number of SSE of P). A vector is associated to an SSE q of the candidate protein Q and its ith element is the score of q with the ith SSE of P. A column vector is inserted into the column vector list only if there is at least a triplet of SSE containing q that is equivalent to one triplet of P. The order of the columns in the linked list representation of the matrix follows the sequential order of the SSE along the backbone chain. This is a requirement of the DP algorithm. The DP algorithm, in its initial version, needed some additional information computed from the internal vectorial representation of the proteins. In the new version this information is no longer needed because the angles and distances of segments are not recomputed during the DP step, but are obtained from the hash table. This results in a further computational advantage.

The modified PROuST can be summarized as follows:

1. **Pre-processing phase:**
   - Hash-table construction and updates.
2. **Index-based structural alignment:**
   - Search for similarities of the target protein with all proteins stored in the hash tables and build the score matrices.
   - Use the DP algorithm to obtain the optimal path in each score matrix and the corresponding optimal structural alignment.
3. **Atomic pairwise protein superimposition:**
   - Using Horn's algorithm, on user's demand.

As before, the pre-processing phase is executed off-line, locally on a cluster of machines. Each machine of the cluster creates a sub-table of the hash table, containing a subset of the PDB proteins. The input proteins are inserted into the sub-tables by a greedy procedure that randomly partitions the proteins into groups of fixed size k, with k larger than the number of sub-tables, and assigns each group to the first available computer of the cluster for insertion into its associated sub-table. Our experimental results show that this simple procedure generates sub-tables of approximatly the same size; furthermore it distributes the set of all proteins almost uniformly across the sub-tables. The pre-processing phase is also triggered by updates in PDB, or by timeout expiration (each month), or by user intervention. The last phase 3 is executed locally, on-line, and takes in input a pair of proteins, on user's demand. Only the index-based structural alignment is executed on-line on the nodes of a computational grid. In the distributed master-slave implementation each slave performs index based structural alignement of the target protein with proteins stored into its associated hash sub-table.

4. Gridifying PROuST

In the previous section, we have shown how to restructure our application, envisaging its use on a computational grid. Here we explain how to exploit the grid capabilities to optimize the execution time and data storage. We discuss data replication strategies proposed and adopted in porting our application on a grid [7, 10].

4.1 Data distribution and replica

Replica optimization is a crucial aspect of a gridifying strategy, because the availability of data in Storage Elements (SE) close to the Computing Element (CE) where the job requesting the data runs enables latency reduction and efficient access. Replica management in the European DataGrid and Grid.it is handled by independent services interacting via the Replica Manager (RM) [7]. In a grid, a file is first registered with an identifier, the Grid Unique ID (GUID), then it is replicated and distributed. The main advantage of using replicas is that one can refer to a file simply using its GUID, then the RM through its *Replica Catalog* (RC), links the GUID to all the replicas of the file. Referring to all replica files using a unique GUID allows to ignore how many replicas are in the grid and where they are. The *Replica Optimization Service* (ROS) selects the best available replica of the data files a job needs. Using these services we experimented with two main policies for replica management, that we called *on-line replica* and *off-line replica*.

On-line replica means that every time a job is executed on a CE, the Replica Manager creates a replica of all data files the job needs in the SE placed at the same site, if they are not already available there. This strategy implies that the user has high control over the storage resources. In fact, replicating data files on-line is possible only with a permission to insert new files in an SE of the Virtual Organization (VO) and to delete local replica of other files if no sufficient space is available. This permission is not always granted because, typically, a single storage resource is shared by many users and even by many VOs. Moreover this strategy is beneficial only when the time spent accessing data from a remote site is greater than the overall time needed to replicate data files at the local site and access them locally.

The *off-line replica* policy refers to the case when all possible replica operations are made once and for all in a pre-processing phase, before the user submits any job and the scheduler assigns jobs to the CEs. In this scenario, data are always accessed from the SE selected by the ROS taking into account the geographical distribution of the resources and the network latency. This latter strategy seemed to be more suitable to our application on a real grid. The different steps of an application running on a grid environment are summarized in Figure 2.

4.2 PROuST on Grid.it

Our distributed implementation was developed for the Italian Grid.it, more specifically for the INFNGrid that is also part of the project Grid.it [2]. The INFNGrid counts more than 20 sites among Italian institutions. Its main applications were originally in physics, however it has become also open to other fields such as bioinformatics and biomedicine.

Due to the large number of users and VOs involved in this grid project, we adopted an *off-line replica* policy as described in the previous section, by replicating our dataset in all available SEs of the INFNGrid (see Figure 3). We used the Globus default scheduling strategy even if in the future we plan to explore other possibilities. Also we decided to manage the Globus Toolkit and the Replica Manager services using the standard Command Line Interface.

We experimented with different partitions of the hash table into subtables with the aim of minimizing the time spent for data transfers and reducing the overall execution time by increasing the degree of parallelism. Initially, the hash-table was partitioned into three subtables, each of more than 1.5GB. The time spent in downloading the subtables was critical. Dividing the hash-table into a larger number of subtables reduces their dimension from 1,5GB to about 500MB for 9 subtables, and to about 150MB for 30 subtables.

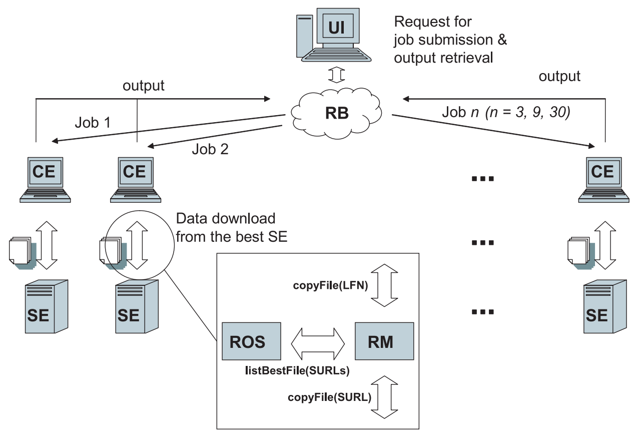

Figure 2. Application execution steps in a grid environment.

Time analysis for data transfers and application's execution times is shown in the next section.

5. Time analysis

Before running the application on a grid, we conducted experiments to evaluate the effect on time performance of the algorithmic changes to PROuST discussed in section 3 when it is executed on a single computing element, a standard PC. In the following, versionB and versionA are the two versions of the algorithm, before and after the re-organization of its components, respectively. Execution times are determined for three partitions of the hash table, into 3, 9 and 30 subtables. The execution time breakdown for versionB are reported in Table 1: t_1 refers to the average time to obtain from an hash subtable the list of candidate proteins. This time, for a given partition of the hash table, is averaged over all subtables. It has to be noted, however, that there is little variance across the subtables. t_2 is the total execution times of all DP computations to align the target protein with the candidate proteins extracted from a subtable; this time is averaged over all subtables of the hash table partition. Recall that, in versionB, the DP algorithm builds the score matrix and re-computes the angles and distances of all triplets of secondary structures of the target and of the candidate protein. The execution times in Table 1 were obtained with the input protein 1a2z (chain C), a peptidase protein, that contains 14 SSE. Only 12 SSE were included in our analysis since SSE with less than 3 residues were discarded.

time (sec.)	**3 sbts**	**9 sbts**	**30 sbts**
t_1*: index-based search*	19	5	1
t_2*: DP + score matrices building*	1016	370	136

Table 1. VersionB (before re-organization). Execution time breakdown for protein 1a2z chain C on a single CE.

time (sec.)	**3 sbts**	**9 sbts**	**30 sbts**
t_3*: index-based search + score matrices building*	980	84	19
t_4*: DP*	2	< 0.5	< 0.5

Table 2. VersionA (after re-organization). Execution time breakdown for protein 1a2z chain C on a single CE.

Table 2 shows the execution times of the re-organized components when executed with the same target protein 1a2z as input. In versionA, the hash table search includes the building of the score matrices (t_3), thus the DP (t_4) does not have to re-compute the geometric properties of SSE. As can be seen from these results, versionA is consistently better than versionB and the gain in performance increases with the number of subtables. Furthermore, results on proteins of different sizes, i.e. different number of SSE, show that this improvement is more relevant for proteins of average/small sizes. For very large proteins, relatively infrequent in the database PDB, some improvement can still be observed but it is negligible.

These experiments, repeated over a large set of proteins, led us to the conclusion that the algorithmic changes introduced in this paper result in an enhancement even when the application was executed an a sin-

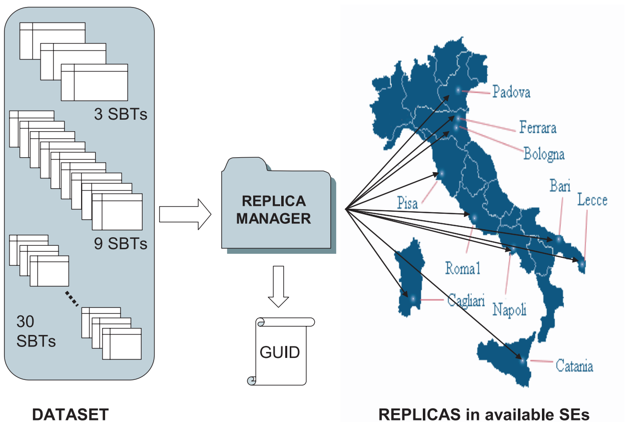

Figure 3. Data Replication.

gle CE. Thus we can only expect a further improvement when porting the application on a grid. Based on these premises, we proceeded to the implementation of the new version of PROuST on Grid.it.

The time analysis is divided in two parts: the analysis of the time spent to download the subtables and of the overall execution time. We have to make some preliminary remarks: the analysis of the time needed to download the data and perform the computation involves many variables that are hardly predictable. Among them, the most crucial is the network traffic that can slow down the speed of downloads significantly. Also important is the load of the grid's nodes that can severely affect the time performance. In a real grid environment, we observed variations of the execution times for different runs of the same task even by one order of magnitude. To overcome these problems, we run the application several times under different conditions of network traffic and performance of the nodes. The times reported in the following are the mean values over all these runs. Our tests were made during a period of about two months, from August to October 2005, to ensure a good variability in terms of grid conditions. We determined the time to download an hash subtable for the three different partitions of the original table into 3, 9, and 30 subtables. For each partition we consider two data distributions:

1. a single SE stores all subtables,
2. all subtables are replicated in the grid SEs;

Table 3 summarizes the results. As expected, the increased granularity in table's subdivision and the data replica throughout the storage elements positively affect the data transfer time. We show below the execution times of PROuST for sixteen input proteins, with sizes ranging from 5 to 99 secondary structures. These times are for the overall computation: the index-based search and all DP computations to align the target protein with all candidate proteins. The overall execution time benefits by the increased number of subtables, up to 30 subtables, from which point on we could observe a degradation in time performance. Execution times for each protein chain are summarized in Table 4 and Figure 4. In Figure 4 times are presented in *log* scale. From this figure we observed that by increasing the number of subtables and therefore the parallelism degree we obtain a constant ratio of the execution times. In fact if let T_1 and T_2 be the execution times obtained for the same protein chain with different hash table partitions, with $T_2 > T_1$, we have:

time (sec.)	**3 sbts**	**9 sbts**	**30 sbts**
Single SE	748	279	90
Many SEs	492	160	49

Table 3. Data Transfer Time.

$$\log T_2 - \log T_1 = \log \frac{T_2}{T_1} = c \Rightarrow \frac{T_2}{T_1} = K$$

with both c and K constant values.

The software package consists of a C program for comparing an input protein against all proteins of the PDB, and of a pool of BASH scripts to manage jobs description, submission and status control over the grid.

6 Conclusions

We have presented a distributed implementation on Grid.it of a software tool for protein structure comparison. A major restructuring of the application has been

Protein	#SSs	3 sbts	9 sbts	30 sbts
4hck .	5	17 (s)	4 (s)	1 (s)
110m .	8	263	32	7
111m .	8	228	31	8
112m .	8	221	31	8
1etc .	8	199	21	4
1fgz A	8	180	20	4
2gva A	9	206	26	6
2gva B	9	228	26	5
1a2z A	12	1098	99	17
1a2z B	12	1046	99	18
1a2z C	12	980	88	16
1a2z D	12	1009	90	16
9xia .	20	4238	476	87
8icm A	22	4874	561	102
1ea0 A	95	N/A	4986	676
1ea0 B	99	N/A	5279	707

Table 4. PROuST Execution Times.

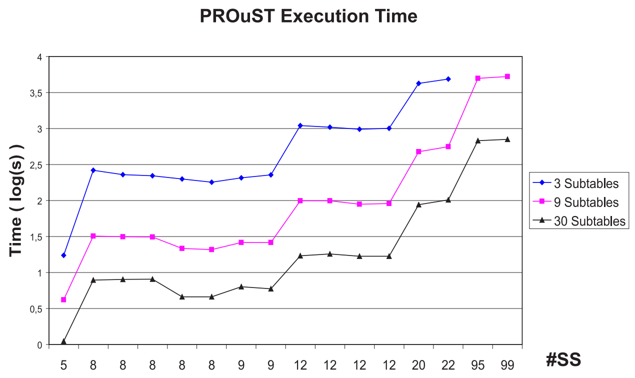

Figure 4. PROuST Execution Time.

developed for an efficient porting on a computational grid. Without restructuring the DP phase of the application makes many small requests of data that involve frequent and costly accesses to SE. The re-organized version of the software uses the data available from the hash tables for more than one operation once they are accessed; i.e. for the determination of the list of candidate proteins and also for the computation of the score matrices used by DP. Furthermore, by re-using the hash table data we avoid the access to the many small files containing the protein vectorial representations that would be needed by DP if the geometric properties were to be recomputed. We have also experimented with different ways of exploiting data replica on the grid, and with different partitions of the hash table into sub-tables. The algorithmic changes and the gridification strategies employed allow for a significant reduction in communication time and overall execution time.

References

[1] The Protein Data Bank, Research Collaboratory for Structural Bioinformatics (RCSB). www.rcsb.org/pdb

[2] M. Aldinucci, S. Campa, M. Coppola, M. Danelutto, D. Laforenza, D. Puppin, L. Scarponi, M. Vanneschi and C. Zoccolo, Components for high performance programming in the Grid.it project. *Proc. of Intl. Workshop on Component Models and Systems for Grid Applications*, ACM ICS, 2004.

[3] M. Comin, C. Guerra, G. Zanotti, PROuST: a Comparison Method of Three-Dimensional Structure of Proteins using Indexing Techniques. *Journal of Computational Biology*, Vol. 11, No. 6, pp 1061-1072, 2004.

[4] M. Comin, C. Ferrari, C. Guerra, Grid deployment of bioinformatics applications: a case study in protein similarity determination. *Parallel Processing Letters*, Vol. 14, No. 2, pp 163-176, 2004.

[5] I. Foster and C. Kesselman, The Grid: Blueprint for a Future Computing Infrastructure. *Morgan Kaufmann Publishers*, 1999.

[6] C. Ferrari, C. Guerra, G. Zanotti A grid-aware approach to protein structure comparison. *J. of Parallel and Distributed Computing*, Vol. 63, No. 7-8, pp 728-737, 2003.

[7] D. Cameron, J. Casey, L. Guy, P. Kunszt, S. Lematre, G. McCance, H. Stockinger, K. Stockinger, G. Andronico, W. Bell, I. Ben-Akiva, D. Bosio, R. Chytracek, A. Domenici, F. Donno, W. Hoschek, E. Laure, L. Lucio, P. Millar, L. Salconi, B. Segal and M. Silander, Replica Management in the European DataGrid Project. *Journal of Grid Computing*, Vol. 2, No. 4, pp 341-351, 2004.

[8] J.G. Jensen, T. Shah, O. Synge, J. Gordon, G. Johnson and R. Tam, Enabling Grid Access to Mass Storage: Architecture and Design of the EDG Storage Elements. *Journal of Grid Computing*, Vol. 3, No. 1-2, pp 101-112, 2005.

[9] J. Montagnat, F. Bellet, H. Benoit-Cattin, V. Breton, L.Brunie, H. Duque, Y. Legré, I.E. Magnin, L. Maigne, S.Miguet, J.M. Pierson, L. Seitz and T. Tweed, Medical Image Simulation, Storage, and Processing on the European DataGrid Testbed. *Journal of Grid Computing*, Vol. 2, No. 4, pp 387-400, 2004.

[10] D. G. Cameron, A. Fr. Millar, C. Nicholson, R. Carvajal-Schiaffino, K. Stockinger and F. Zini, Analysis of Scheduling and Replica Optimization Strategies for Data Grids Using OptorSim. *Journal of Grid Computing*, Vol. 2, No. 1, pp 57-69, 2004.