Grid Deployment of Legacy Bioinformatics Applications with Transparent Data Access

Christophe Blanchet[#1], Rémi Mollon[#2], Douglas Thain[*3], Gilbert Deléage[#4]

[#] *Institut de Biologie et Chimie des Protéines*
IBCP UMR 5086;CNRS; Univ. Lyon1;IFR128 BioSciences Lyon-Gerland;7, passage du Vercors, 69367 Lyon cedex 07, France
[1]Christophe Blanchet@ibcp.fr
[3]Remi.Mollon@ibcp.fr
[4]Gilbert.Deleage@ibcp.fr

[*] *Department of Computer Science and Engineering, University of Notre Dame,*
384 Fitzpatrick Hall, Notre Dame, Indiana, United States
[2]dthain@cse.nd.edu

Abstract— **Although grid computing offers great potential for executing large-scale bioinformatics applications, practical deployment is constrained by legacy interfaces. Most widely deployed bioinformatics were designed long before grid computing arose, and thus are created, tested, and validated in the familiar environment of a workstation. Most perform simple local I/O and have no facility for interfacing with a distributed system. Because of these limitations, users of bioinformatics applications are generally constrained to creating large local clustered systems in order to perform data analysis. In order to deploy these applications in wide-area grid systems, users require a transparent mechanism of attaching legacy interfaces to grid I/O systems. We have explored this problem by deploying several bioinformatics databases and programs for protein sequence analysis on the European EGEE grid. Using tools for transparent adaptation, we have connected legacy applications to the logical namespace provided by a replica manager, and compared the performance of remote access versus file staging. For common bioinformatics applications, we find that remote access has performance equal or better than simple file staging, with the added advantage that users are freed from stating the data needs of applications in advance.**

I. Introduction

Managing the deluge of biological data produced by large-scale experiments such as genome projects is one of the major global challenges of bioinformatics. Biological datasets present challenges from many different directions. There are many laboratories that produce data that must be archived, distributed, and indexed. There are many consumers of data that must identify, search, and manipulate data sets. There are many different resource stakeholders that place constraints on how networks, storage, and processors may be used, and who may use them. Most importantly, the scale of data to be processed requires access to large collections of widely distributed resources such as found in computational grids.

The analysis of this deluge of data will need to be farmed out to large computing resources with large and efficient storage devices. Indeed, bioinformaticians need for their daily analyses of genomes efficient access to these biological data and bioinformatics programs. However, these datasets are not static: as discoveries are made, new entries are added to the database and existing ones are updated. For example, the Swiss-Prot database is updated biweekly as a new complete database file containing all the entries in their last release, but can be updated every day as a file containing only the updated entries. As these updates occur, they must be propagated to the various copies spread around the grid, and perhaps re-executing programs after the updates take place.

Complicating the problem of data management is the fact that most bioinformatics applications are not designed with grid computing in mind. Indeed, many valuable applications were designed, tested, and validated long before grid computing arose. As a result, such applications are designed to perform simple local I/O and have no facility for attaching to grid data systems. Regardless, these applications need both high-throughput computing and huge data storage [6][14].

Why not simply re-write such applications to take advantage of the grid? Although this might be possible for a small number of applications, re-writing would be an enormous amount of work to address all bioinformatics codes in use today. Many are commercial codes for which source is not available. In some cases, the construction and validation of codes is tightly integrated into an audited scientific process; changing the code for grid deployment would invalidate the application, or at least require a re-certification. To make grid computing easy, we must find a way to access data through familiar interfaces without changing applications.

In this paper we present how grid computing could be a viable solution to distribute and integrate large bioinformatics databases, and to make these distributed databases usable by legacy bioinformatics programs. We employ the European EGEE data management system for managing, replicating, and locating large database files. We use Parrot [22] to connect legacy applications to the EGEE data management system. We modify Parrot to support a logical name space, thus freeing users from managing (or even seeing) physical file locations. Finally, we evaluate these tools on a set of representative biological databases and bioinformatics programs for protein sequences analysis. We find that the combined system provides performance equal or better than simple file staging, with the added advantage that users are freed from stating the data needs of applications in advance.

II. BIOINFORMATICS AND PROTEIN SEQUENCES ANALYSIS

A. *Biological Data*

Biological data are available from many different locations around the world. They are very large datasets of different natures, from different sources, structured according to heterogeneous models (See Table I). Genome projects produce large sets of raw sequence data needing to be analyzed and annotated in order to be useful to the scientific community [16]. Biological data are also produced by other experimental techniques (biophysics, molecular biology, biochemistry.), and stored together in as many databases of different nature: protein three-dimensional structures, functional signatures, expression arrays, etc. Storing and analyzing these data require translation into different representation of data such as (i) alphabetical for genes and proteins, (ii) numerical, for structural data from X-ray crystallography or NMR, or (iii) image for 2D-gel. There are currently more than 700 public biomolecular databases [11] (see some examples in Table I). All these data are distributed, analyzed, crosschecked to other databases, used to predict new data, and published into scientific journals. The ratio between high-level and raw biological data is generally estimate to be a factor of 1,000. Thus, the scale that can be estimated for biological data is around petabytes.

TABLE I
SIZE OF SOME BIOLOGICAL DATABASES

Name	**Nature**	**Rev.**	**Entries**	**Size (MB)**
GenBank	Gene Sequence	153	56,620,500	224,000
EMBL	Gene Sequence	86	69,783,593	~100,000
Swiss-Prot	Protein Sequence	49.5	216,380	824
TrEMBL	Protein Sequence	32.5	2,807,081	6,347
PROSITE	Protein Signature	19.25	1,411	14
pFAM-A	Protein Signature	19.0	8,183	2,104
PDB	Protein Structure	Apr. 2006	36,121	23,316

Most of these databases record their data with their own specific format into flat text files. The data are generally split into textual entries composed of key-value pairs, which are centered on a datum to which are associated metadata. In a protein sequence dataset, the datum is the sequence, and metadata are items such as the identifier of the protein, the bibliographic references of works on the protein, and the references to structures published into the Protein Data Bank. These data can be scanned in a goal of biological data mining or analysis through suitable Web form on their own portal. These worldwide databases are available on Internet through for example HTTP or FTP transfers for a desktop or site usage [4][7][16]. In this case biologist has to get one of several files of different sizes containing full release or release updates (see Table II).

Moreover, biological databases are not static. Every day, new data are published and existing ones may be updated. Thus, biological databases need to be periodically updated. Maintaining several instances of these databases becomes rapidly a nightmare for end-user as biologist. Not only must they have a mechanism for obtaining and updating local copies, they must also maintain a mapping from the logical identity of the data to one or more local copies. On top of, most worldwide databanks like Swiss-Prot [4], TrEMBL [4], GenBank [5] or EMBL [20] have doubled their volume each year. Soon, it will not be practical for every biologist to store a local copy.

TABLE II
EXAMPLE OF BIOINFORMATICS PROGRAMS

Name	**Algorithm**	**Input data**
BLAST	Similarity	Gene/Protein Sequence
FASTA	Similarity	Gene/Protein Sequence
SSearch	Similarity	Gene/Protein Sequence
ClustalW	MSA	Protein Sequence
Multalin	MSA	Protein Sequence
PattInProt	Pattern/Profile	Sequence, Pattern, profile
GOR4	PSSP	Protein Sequence
SIMPA96	PSSP	Protein Sequence
SOPMA	PSSP	Protein Sequence

PSSP: Protein secondary structure prediction; MSA: Multiple Sequence Alignment

B. *Bioinformatics applications and I/O requirements*

In Bioinformatics as in other disciplines, scientists are using and producing very different and numerous methods to analyze their data. For each scientific domain of Bioinformatics, they are very often several different high-quality programs available for computing the same dataset in as many ways. For instance, the BioCatalog [29] was referencing at the end of 1990s all the bioinformatics programs available to the community: more than 600 had been registered into this database, and most probably as many being not. Another example is the EMBOSS software suite [17], comprising more than 200 programs devoted to molecular bioinformatics. Each of these programs answer to as many different biological queries, from very simple as converting data format to more complex ones such as querying for sequence similarity between one sequence and a whole database.

The subject of protein sequence analysis concerns the analysis of newly discovered against known sequences: new ones are obtain for example by sequencing project and known ones are registered within protein databases such as Swiss-Prot, TrEMBL, PIR [26] or “non-redundant” from NCBI. Protein sequence algorithms are also computing different kind of input data than sequences: for instance functional sites and signatures (represented as patterns or HMM profiles [13]), or protein 3D structures. These algorithms can be classified into different domains, according to the data they are analyzing and the results they are producing: similarity search, sequence multiple alignment or pattern/profile scans, etc. (see Table II). Several programs are very well known and largely used by the bioinformatics community, for instance, BLAST [1], SSearch [15], FastA [19], or ClustalW [24].

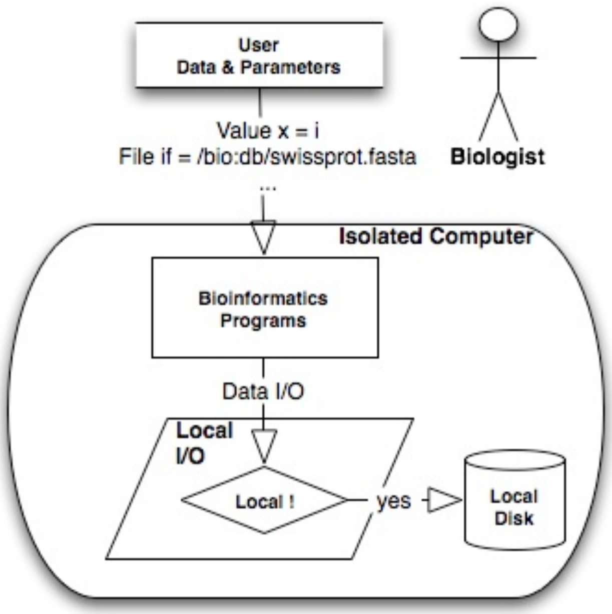

Fig. 1. I/O Model of legacy protein sequence analysis programs. Most of these programs are only able to access to local data.

Most of these bioinformatics programs are not adapted to distributed platform. Indeed, they have been developed in most case in a time where only local computing was the usual way to compute, at least in the bioinformatics community. One main issue is certainly that they are only able to access and store data through local I/O. In Figure 1, a simple model, centered on the program, describe the I/O of these legacy bioinformatics programs. Data processed during the computation could be provided by the user or extracted from databanks such as these described in Table II. The input data can be (i) one or more parameters, (ii) user data, and (iii) databases. They are submitted to the program through the standard command line interface or through other interface, user-friendlier, such as Web pages. The main differences between user data and databases are their size and their location on the given distributed platform, coupled to the fact that the database is not modifiable by the user, but only by the responsible scientific publisher. This difference implies that (i) user data can been transferred at job submission from the user interface to the remote computing node; (ii) databases must be registered and replicated in the storage area of the distributed platform, identified with logical name within a defined namespace, and then transferred from the storage area to the computing nodes.

C. *Bioinformatics representative resources*

Among the numerous bioinformatics programs we have chosen several representatives. We have taken them as models along this works, because of their special requirements for input or output of data. The selected programs have been BLAST, FastA, SSearch, and the associated databases Swiss-Prot [4] or TrEMBL [4].

BLAST, FastA and SSearch are representative of bioinformatics applications having requirement that must access large biological databases (as described before) as reference sets to compute their analyses, for instance protein sequences or pattern databases. They also produce large outputs that can be a subset of the input database. This database may then be pipelined to another bioinformatics program.

BLAST will be used as a model for the class of bioinformatics programs that have requirement of implicit parameters files in input. When an analysis is launched with BLAST, user has to put on the command line, the pathway of the database of protein sequences that will be compared to the unknown sequence. These index files have the same filename and pathway than the database file except for the extension. Thus the extension of the sequence file are most time “.seq” or “.fasta”, and their index files are suffixed with the following three extension “.phr”, “.pin” and “.psq”, according to the three kind of indexes needed by BLAST. But these index filename are never put on the command line, they are implicitly expected by the program BLAST

III. DISTRIBUTED STORAGE AND COMPUTING: THE GRID

A. *An example of Grid: the European EGEE grid*

Grid computing concept defines a set of information resources (computers, databases, networks, instruments, etc.) that are integrated to provide users with tools and applications that treat those resources as components within a « virtual » system [10][23][25]. Grid middleware provides the underlying mechanisms necessary to create such systems, including authentication and authorization, resource discovery, network connections, and other kind of components.

The Enabling Grids for E-sciencE project (EGEE [32]), funded by the European Commission, aims to build on recent advances in grid technology and to develop a service grid infrastructure. The EGEE consortium involves 70 leading institutions in 27 countries, federated in regional Grids, with currently a combined capacity of 20,000 CPUs and 5 petabytes of storage. The platform is built on the LCG-2 middleware, inherited from the EDG middleware developed by the European DataGrid Project [34] (EDG, FP5 2001-2003). The middleware LCG-2 is based upon the Globus toolkit release 2 (GT2) [35] and the Condor middleware [23].

The new middleware gLite [32], that is being developed, have the goals to improve the performances and the services provided by the future EGEE platform.

There are several important components into the EGEE grid: first on the user point of view is the user interface (UI) where the user log in and submit their jobs. These jobs need to be described by JDL files (Job Description Language) with the Condor “ClassAd” formalism. The “workload management system” (WMS) is responsible of the job scheduling on the platform. The scheduler (or “resource broker”, RB) analyzes the JDL file and determines where and when to compute a job: (i) using one “computing element” (CE) near one “storage element” (SE) containing the data in case of simple jobs, or (ii) several CEs and SEs in case of larger jobs. A computing element is a gatekeeper to a cluster of several CPUs, the worker nodes (WN) managed by a batch scheduler system. The “information system” that centralize all parameters raised by the grid components (CPUs, storage, network, …).

In this paper, we are most concerned with the EGEE data management system (DMS), which is responsible for replicating locating files across the grid.

B. Distributed storage on the EGEE grid

The “data management system” (DMS) on the EGEE is composed of several components. The real element doing the storage is the “storage element” (SE), which stores the file on different kind of medium: disk or tape. A SE is most of time attached to a given computing element. We then call this SE the “near-SE” of this given CE. Other components participating to the replica manager system are the replica catalog and a set of commands available from both the end-user interface as well as individual worker nodes. Users can explicitly manage data with different commands: for instance files may be registered into the system, replicated to other sites, or deleted from the system entirely.

Within the DMS, files are available through two namespaces: one logical and one physical. Logical File Names (LFNs) are globally unique and describe a file by its issuing authority and abstract properties. Storage File Names (SFNs) describe the current physical location of a file. A vital job of the DMS is to mapping an LFN to one or more SFNs.

For example, the LFN for a file issued by the author’s institution might be:

lfn://ibcp.fr/2006/proteins.data

To find a copy of this file, a user must contact the DMS in order to obtain a list of SFNs. Each SFN may indicate an FTP server where the file may be found, perhaps under an entirely different name: Possible SFNs for this file might be:

gridftp://ftp.ibcp.fr/data/p2006.data
gridftp://ftp.nd.edu/ibcp/proteins.data

Despite these tools to get the SFN from LFN, there are no automatic substitution mechanisms for applications to employ. A legacy bioinformatics application launched on the EGEE grid won’t be able to access the remote data stored on SEs, unless the user first resolves the LFN to an SFN, and the copies the file from the storage element to the given worker node before executing the program. The DMS on the EGEE grid is a key service for our bioinformatics applications. Having efficient usage of it will be synonymous of good distribution of our protein sequence analysis applications.

C. Parrot: Custom I/O Services for Legacy Applications

In conventional systems, new I/O services have been attached to applications by either modifying the operating system or by modifying the application itself. For example, traditional distributed file systems are implemented in privileged file system drivers, while new grid file access modes are provided by entirely new interfaces. Neither of these modes is acceptable for bioinformatics applications on EGEE: the kernel cannot be modified on execution nodes, nor can we expect end users to modify applications.

Parrot [22] addresses this problem by allowing novel I/O services to be transparently connected to unmodified applications without requiring kernel changes or any privileged activity. Parrot operates by running an application as a child and trapping all attempted system calls using the debugging interface. Modules within Parrot give the application access to a variety of external I/O protocols. Any I/O service implemented as a C library can be connected to Parrot by writing an appropriate module. Each I/O protocol is presented as a directory in the file system namespace. For example, a GridFTP server named ftp.ibcp.fr may be accessed under the path /gridftp/ftp.ibcp.fr.

As discussed above, most applications do not wish to address data based on the physical name. Rather, they wish to access a logical address that is then applied to an underlying name. Parrot provides two mechanisms for making this mapping at runtime: a static mount table, and a dynamic name resolution hook.

The **static mount table** resembles the file system mount table in a conventional operating system. It consists of a series of entries mapping a logical file name to a storage element. For example, suppose a BLAST application expects to find its data in the /db directory and configuration files in the /home/blast directory. These logical names can be directed to two different GridFTP servers with the following mount table:

/db	**/gridftp/ftp.ibcp.fr/data**
/home/blast	**/gridftp/ftp.nd.edu/home**

This mechanism can be applied when the set of files needed is known in advance and those in the same logical directory are stored in the same physical location. However, for more complex configurations, a dynamic mechanism is needed.

The **dynamic name resolution hook** is an internal routine of Parrot called every time a file is accessed by name. This gives the opportunity for a module to redirect file requests for any single file at runtime. In particular, this mechanism can be used to resolve logical file names to physical locations using the EGEE file location service. This allows the decision of what file replica to access to be deferred until the moment of access.

IV. RELATED WORK

They have been numerous groups or projects working on grid computing applied to Bioinformatics: Biogrid project [27] in Japan, BRIDGES [30] and eScience [18], EuroGRID [33] in Europe; in US, there are also the Biomedical Informatics Research Network (BIRN) [28] and the North Carolina Bioinformatics Grid [37]. Most of them are deploying their application on experimental grid (Biogrid, myGrid and EuroGRID), or on local cluster (North Carolina). Also large-scale grids are devoting part of their multiple application resources to the bioinformatics applications, like TeraGrid [39] in US, DEISA [31] and EGEE [32] in Europe.

Data are stored and managed in different ways within these platforms. Most of them are managing files, replicating them and providing remote access with new I/O API: LCG middleware with GFAL library in EGEE production platform, gLite with gLiteIO in pre-production platform. Others examples are about the Internet Backplane Protocol (IBP) [3] and the Logistical Runtime System (LoRS) [36]. They supply a C API that simplify and automate the management of exNode (split of file) on the Logistical Bone (LBone), providing users with a networked RAID storage area. Others distributed storage environments are dealing with both files and object stored into DBMS, like the Storage Resource Broker (SRB) [2], OGSA-DAI [38] mediator components, or DiscoveryLink [12]. Some of them are providing application users with mediation interfaces using SQL or XML query interfaces, others through Web services.

TABLE III
SOME BIOINFORMATICS RESOURCES DEPLOYED ON THE EGEE GRID

Resource	**Grid Descriptor**
Swiss-Prot	lfn://genomics_gpsa/db/swissprot/swissprot.fasta
And its indexes in the BLAST format	lfn://genomics_gpsa/db/swissprot/swissprot.fasta.phr lfn://genomics_gpsa/db/swissprot/swissprot.fasta.pin lfn://genomics_gpsa/db/swissprot/swissprot.fasta.psq
TrEMBL	lfn://genomics_gpsa/db/trembl/trembl.fasta
PROSITE	lfn://genomics_gpsa/db/prosite/prosite.dat lfn://genomics_gpsa/db/prosite/prosite.doc
ClustalW	ESM tag “genomics_gpsa_clustalw”
SSearch	ESM tag “genomics_gpsa_ssearch”

Database files have been registered with logical filenames (LFN) within the Data Management System (DMS), and programs have been deployed on sites, registered also with logical names within the experiment software manager (ESM tag).

The HandleSystem [40] is also providing unique persistent identifiers for Internet Resources, but these services are linked to license fees that could be a limitation. Moreover the combinaison of prefix and suffix is making a handle composed of numeric string that is not human-friendly, for example, "4263537/4000" is a handle for the Handle System web site home page. A easy-to-understand naming space is needed as the DNS is providing us with IP addresses resolution. The GLARE [41] system is a Grid-level application that provides users with component registration, deployment and provisioning framework. This service is focusing on grid applications (activities) deployment, but is not dealing with remote access to databases by legacy applications that are only able to do local file access calls.

All of these related works are assuming that users either modify the bioinformatics application codes, or that they replicate the data on the given nodes before the planned computations. Modifying the whole bioinformatics programs implies to include the appropriate library calls (in C, Java or others) in them, in order to enable them to do remote access calls to the remotely-stored biological databases.

V. DEPLOYING BIOINFORMATICS RESOURCES WITHIN EGEE

A. Virtualization of Bioinformatics Resources

1) Defining a Bioinformatics Namespace

Despite the replica and the software management system have no file tree hierarchy, we have decided to create our specific namespace with chosen logical filenames and program tags. We have then deployed on the EGEE grid the representative biological databases and bioinformatics programs selected above (see Table III). The data files have been registered in the EGEE replica management service (RMS) with the appropriate LFNs; and bioinformatics programs have been registered tags in the experiment software management service (ESM) on several computing elements:

- databases: lfn://genomics_gpsa/db/dbname/dbfiles
- programs: genomics_gpsa_program

Both the deployed LFNs and ESM tags play a key-role into the scheduling of our future jobs. Indeed, jobs submitted with LFN and/or ESM tags within the user’s submission file, will be sent according to matchmaking between these logical names and the free nodes hosting one physical replica of them. We assume that the files have already been replicated to several storage elements on the grid according to external constraints.

2) Virtualizing remote I/O on EGEE

We have adapted Parrot tool for our application usage on the EGEE middleware, we called this modified version “Perroquet”. The main change is about the file namespace understood by Parrot. We have added the code for the recognition of true URL as the LFN is:

- <protocol>://<hostname>/path/to/resource
- lfn://genomics_gpsa/db/swissprot/swissprot.fasta

When have also added in Parrot the mechanism for resolving these LFNs by querying the EGEE file catalog that is in charge of making the name resolution between logical namespace and real filenames (SFNs) on the distributed storage elements.

Perroquet is then able to identify LFNs among the program command line arguments, to resolve names to locations (SFNs) and to get the corresponding SFN from a SE (see Figure 2). Thanks to Parrot source code, several transfer protocols are available such as for instance FTP, HTTP, or GSI-FTP. We use this last one because it is authenticated, encrypted and available from the EGEE storage elements.

The remote I/O are done with local disk cache or without for security purpose. In local disk cache mode, Perroquet puts the relevant access rights on the cache file and directory and remove them after the run. In “on-the-fly” access mode, Perroquet is caching remote I/O in a 64KB memory buffer.

B. Legacy application without remote I/O

As legacy bioinformatics applications have no remote I/O to access remote databases, we need to encapsulate them into an execution space that forward their local I/O to remote locations. We have tested two ways of providing them with remote access to biological database: file replication on the local computer or remote I/O. They both mean that the prerequisite databases have to been downloaded from the remote grid and pushed to the input stream of the program. As we cannot modify the application, this remote access has to be done by an agent separate from the bioinformatics program..

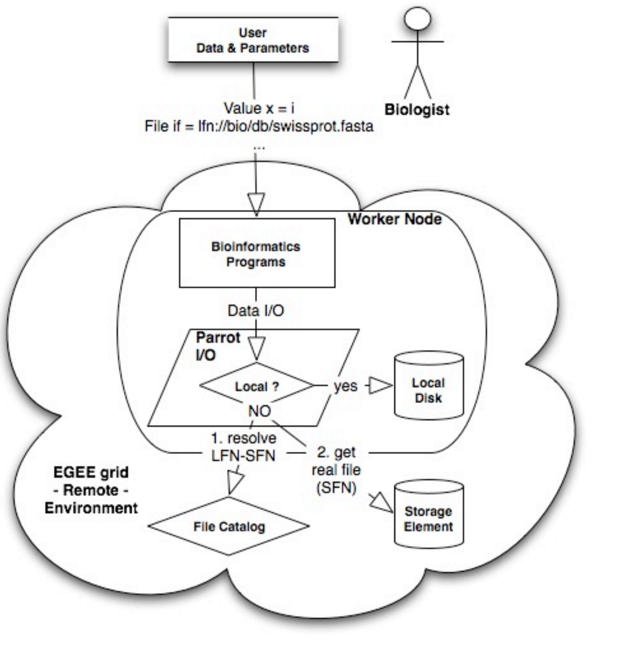

Fig. 2. Parrot integration into the EGEE platform. Architecture of the data management system with Perroquet, and its integration into the EGEE grid platform and middleware (UI: user interface, CE: computing element, SE: storage element, WN: worker node, DB: database, LFN: logical file name). Perroquet is catching the local I/O and forwarding them through network to remote storage element. Several transfer protocols are available such as FTP, HTTP, and especially GSI-FTP that is authenticated, encrypted and fully compatible with the EGEE grid middleware, component and services. Perroquet, is our adaptation of Parrot to the EGEE grid middleware. It recognizes canonical URL and EGEE LFNs , like lfn://genomics_gpsa/db/Swissprot.fasta, and is also capable of data encryption and decryption “on the fly”

This agent must be able (i) to resolve grid filename(s) to locations - from LFN(s) to SFNs - , (ii) to get the data from the best location - with the best protocol - and (iii) to launch the execution of the program against these downloaded data. We use the Perroquet tool as this launcher

C. File replication vs. remote I/O

1) Deployed study sets of biological data

We have split two protein sequences databases, Swiss-Prot and TrEMBL (184,034 entries for Swiss-Prot release 47.2 and 1,779,481 for TrEMBL release 30.2) into several subsets of different protein sequence number The size in bytes of a subset file is correlated to the number of sequences stored in this subset file, by a linear model, thus we use the number of sequence as reference value for our studies. We have deployed all the subsets onto the EGEE grid platform using the data managing system (DMS). We had labeled them with appropriate logical filenames (LFNs, see Table III) into the replica manager system (RMS), and randomly replicated these LFNs on the storage elements of several grid nodes without applying any particular model of replication.

A local execution model of the selected bioinformatics applications has helped us to calibrate these three programs, and to decide which of them and which datasets to test on the grid: from 200,000-seq subset to 1,000,000-seq for FastA and BLAST which are more data intensive, from 50,000-seq to 200,000-seq for SSearch which is more computing intensive.

2) Free storage space on the WN

In case of file replication to the local storage, the agent must make sure that there is enough free space on the local storage area of the computing node. Indeed, biological database may be several hundred megabytes (see Table I). Thus this WN must be able to store this much data on its local disk because bioinformatics programs need to access the whole database in once run. This constraint of free space will be also enhanced in case of a job accessing to several databases, or in case of a WN with multiple CPU, then accepting multiple in the same time, but sharing the same storage space.

In case of remote I/O with Parrot, the data are put directly on the standard input stream of the program, without caching them on the local disk. In some cases of special access (seek, …), a cache can be needed if the protocol is not supporting such special I/O primitives.

For large files, the remote I/O mode has the advantage, because (i) in most times it doesn’t have to worry about the free local space and (ii) this is too late to do this checking of the free space when the job is on the worker node. Indeed, it should be done before, at least at the job scheduling time. Thus the file copy mode implies to modify both the information system (to record the free space on WNs) and the scheduling mechanism (to include this “free space” element when the matching is done between the job requirements and the available WNs) whereas the remote I/O mode works with the current information system and matching workload.

3) Concomitance of data download and job computation

The file copy mode implies that the agent downloaded the database file before the execution of the programs. In the remote I/O mode, Parrot is providing the program with the file blocks progressively. With computing intensive programs, this should have little impact as with the SSearch program. But in case of program that are more data intensive as BLAST and FastA, the impact should not be negligible.

4) Non-declared input and output files

As explained above, some bioinformatics programs have special behavior regarding the file they needed as input, or the files they create as result files. For instance, BLAST program expects at each run to find, in the same location of the database file, also three others index files; or ClustalW produces always a second output file in the same location than the declared output file (sequence alignment file). But these non-declared files are not present within the command line argument of the given program.

When these files are virtualized with LFNs such as we do it with protein sequence databases (see Table III), the program will try to attend on the same way to prerequisite non-declared LFNs. In case of BLAST, that implies to also gridify the index files with the expected LFNs. This could be done easily.

In file copy mode, the agent has to copy these indexes too. The agent has to identify which bioinformatics program will be launched, and to get the non-declared file. That means to bring knowledge about the program within the agent, with, for example, method-specific configuration files. For instance, in case of BLAST execution, the agent has to download also the indexes of the sequence database. In this way it also increases the constraint about free space required on the WN, as seen before.

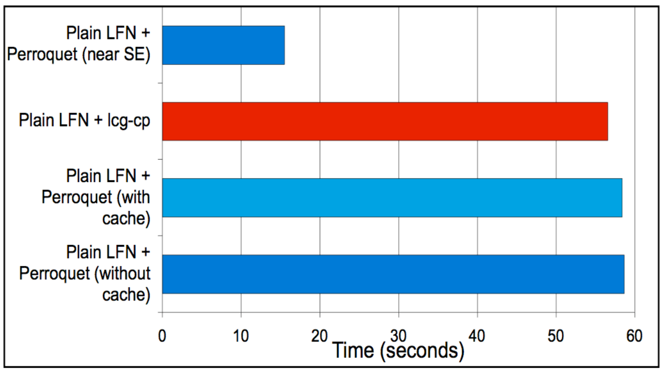

Fig. 3. Download time of a biological database containing 500,000-protein sequence (205 Mbytes). We have compared the access to the database file through the file copy with "lcg-cp" and through remote I/O with the "Perroquet" program. The file is downloaded from any storage element and from the near-SE of the computing element (CE).

The remote I/O mode does not care about such issues. Parrot catches the I/O requests when they are done. When a request is made to open a file the LFN-SFN name resolution is done and the transfer begin from the best location transparently to the user. The same mechanism is played when the program attempts to create a non-declared output file.

5) Performances

In Figure 3 we compare the performances of the two transfer mode, local copy and remote I/O, for accessing to a remote database or 500,000 protein sequences (205 MB), stored on a remote storage element. The database are downloaded from any storage element of the EGEE grid and from the near-SE of the given worker node. These benchmarks show that Parrot has the same efficiency than lcg-cp (~60 sec), both with local cache mode activated or not. This figure also raised the importance of downloading the data from the near-SE of the used WN, and not from any SE of the grid. Indeed, the performances are four times faster in case of access to a SFN on the near-SE. As the biological databases are numerous and have large sizes, we certainly won't be able to replicate all of them on all sites (CE & SE), but we will need to distribute them according to an efficient model of replication

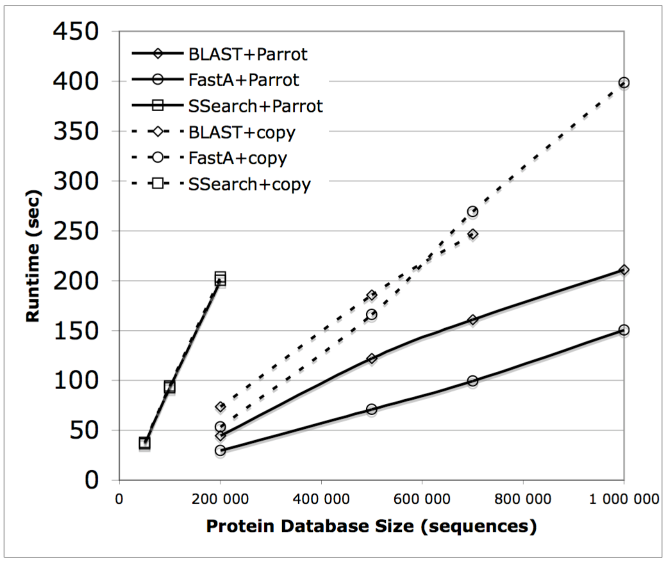

Fig. 4. Comparison of bioinformatics program performances with two different file access modes on EGEE grid. We have been using remote I/O with Perroquet ("Parrot", plain curves) or file copy on the worker node ("copy", dashed curves).

In Figure 4, curves show that bioinformatics program used with Parrot has equal or better execution time than the program computing on copied data. The experiments with lcg-cp show the overhead due to the download of the file before computation. Comparing SSearch curves to BLAST and FastA ones, we see that the overhead is function of the size of the transferred database, and of the used bioinformatics algorithm that can be more or less data intensive, respectively BLAST- FastA and SSearch in our work. As lot of biological database have size larger than 205 MB, the overhead will be in most of cases not negligible for data intensive bioinformatics methods.

VI. Conclusions

We have studied several representative biological databases and legacy bioinformatics programs for protein sequences analysis onto the European EGEE grid. These resources have been virtualized within a devoted namespace, open and available to biologist and bioinformaticians users registered in to the “biomed” virtual organization. Important issues of local-only I/O access have been studied (i) with a local copy of these databases before the computation and (ii) with the remote access provide by the Parrot tool. We observe that Parrot has performance equal or better than file copying, with the added advantage that the user need not declare all files when submitting the job. This grid deployment model of legacy bioinformatics applications have been applied to our bioinformatics workbenches such as the GPSA bioinformatics Web portal, providing biologists with Web transparent access to the EGEE grid.

Future works will be done on the management of access rights and authorizations with Access Control Lists (ACLs) on these encrypted data, because bioinformatics programs are largely used in projects or institutions (like hospitals) that require confidentiality, integrity and authorized access to these distributed biological data.

ACKNOWLEDGMENT

This work was supported in part by the French Centre National de la Recherche Scientifique and by the European Union with the EGEE project (contract number INFSO-508833).

We thank the anonymous reviewers for pointing us to additional related work, and to make this paper clearer .

REFERENCES

[1] Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J., Basic local alignment search tool. J. Mol. Biol. 215 (1990) 403–410.

[2] Baru, C., Moore; R., Rajasekar, A., Wan, M.: The SDSC Storage Resource Broker. Proc. CASCON'98 Conference , Nov.30-Dec.3, 1998, Toronto, Canada.

[3] Bassi, A., Beck, M., Moore, T., Plank, J.S., Swany, M., Wolski, R., Fagg, G. The Internet Backplane Protocol: A Study in Resource Sharing, Future Generation Computing Systems, (19)4, May, pp.551-561. Elsevier.

[4] Bairoch, A, Apweiler, R : The SWISS–PROT protein sequence data bank and its supplement TrEMBL in 1999. Nucleic Acids Res. 27 (1999) 49-54

[5] Benson D.A., Karsch-Mizrachi I, Lipman D.J., Ostell J., Wheeler D.L. ; GenBank: update. Nucleic Acids Res. 32, (2004) 23-26.

[6] Breton, V., Blanchet, C., Legré, Y., Maigne, L. and Montagnat, J. : Grid Technology for Biomedical Applications. M. Daydé et al. (Eds.): VECPAR 2004, Lecture Notes in Computer Science 3402, pp. 204–218, 2005.

[7] Combet, C., Blanchet, C., Geourjon, C. et Deléage, G. : NPS@: Network Protein Sequence Analysis. Tibs, 25 (2000) 147-150.

[8] Desprez, F., Vernois, A., Blanchet, C.: Simultaneous Scheduling of Replication and Computation for Bioinformatic Applications on the Grid. ISBMDA 2005, Lecture Notes in Computer Science 3745: 262-273

[9] Discala, C., Benigni, X., Barillot, E., Vaysseix, G.: DBCAT: a catalog of 500 biological databases. Nucleic Acids Research 28 (2000) 8–9

[10] Foster, I. And Kesselman, C. (eds.) : The Grid 2 : Blueprint for a New Computing Infrastructure, (2004).

[11] Galperin, M.Y.: The Molecular Biology Database Collection: 2005 update. Nucleic Acids Research 33 (2005) National Center for Biotechnology Information and National Library of Medicine and National Institutes of Health.

[12] Goble, C.A., Stevens, R., Ng, G., Bechhofer, S., Paton, N.W., Baker, P.G., Peim, M., Brass, A.: Transparent Access to Multiple Bioinformatics Information Sources. IBM Systems Journal 40 (2001) 532–551

[13] Hulo N., Sigrist C.J.A., Le Saux V., Langendijk-Genevaux P.S., Bordoli L., Gattiker A., De Castro E., Bucher P., Bairoch A. : Recent improvements to the PROSITE database. Nucl. Acids. Res. 32:D134-D137(2004)

[14] Jacq, N., Blanchet, C., Combet, C., Cornillot, E., Duret, L., Kurata, K., Nakamura, H., Silvestre, T., Breton, V. : Grid as a bioinformatics tool. , Parallel Computing, special issue: High-performance parallel bio-computing, Vol. 30, (2004).

[15] Pearson W.R. ; Searching protein sequence libraries: comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms. PNAS (1988) 85:2444-2448

[16] Perriere, G, Combet, C, Penel, S, Blanchet, C, Thioulouse, J, Geourjon, C, Grassot, J, Charavay, C, Gouy, M, Duret, L, Deleage, G..: Integrated databanks access and sequence/structure analysis services at the PBIL. Nucleic Acids Res. 31, (2003) 3393-9.

[17] Rice,P. Longden,I. and Bleasby,A.: EMBOSS: The European Molecular Biology Open Software Suite (2000) Trends in Genetics 16(6) 276--277

[18] Sinnott, R., Atkinson, M., Bayer, M., Berry, D., Dominiczak, A., Ferrier, M.,Gilbert, D., Hanlon, N., Houghton, D., Hunt, E., White, D.: Grid Services Supporting the Usage of Secure Federated, Distributed Biomedical Data. Proceedings of the UK e-Science All Hands Meeting, Nottingham, UK (2004)

[19] Smith T.F., Waterman M.S. : Identification of common molecular subsequences. J. Mol. Biol. (1981) 147:195-197

[20] Stoesser, G, Tuli, MA, Lopez, R, Sterk, P : the EMBL nucleotide sequence database. Nucleic Acids Res. 27 (1999) 18-24.

[21] Thain, D., Klous, S., Wozniak, J., Brenner, P., Striegel, A., Izaguirre, J.: Separating Abstractions from Resources in a Tactical Storage System. Proceedings of Supercomputing 2005

[22] Thain, D. and Livny, M.: Parrot: an application environment for data-intensive computing. Scalable Computing: Practice and Experience 6 (2005) 9-18

[23] Thain, D., Tannenbaum, T. Livny, M.: Distributed computing in practice: the Condor experience. Concurrency and Computation 17 (2005) 323-356.

[24] Thompson, JD, Higgins, DG, Gibson, TJ : CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic Acids Res. 22 (1994) 4673-4680.

[25] Vicat-Blanc Primet, P., d’Anfray, P., Blanchet, C., Chanussot, F. : e-Toile : High Performance Grid Middleware. Proceedings of Cluster’2003 (2003).

[26] C.H. Wu, L.L. Yeh, H. Huang, L. Arminski, J. Castro-Alvear, Y. Chen, Z. Hu, R.S. Ledley, P. Kourtesis, B.E. Suzek, C.R. Vinayaka, J. Zhang, and W.C. Barker. The Protein Information Resource. Nucleic Acids Research, 31: (2003) 345-347.

[27] Biogrid project, www.biogrid.jp

[28] Biomedical Informatics Research Network (BIRN), www.nbirn.net

[29] The BioCatalog, corba.ebi.ac.uk/Biocatalog

[30] BRIDGES project. www.brc.dcs.gla.ac.uk/projects/bridges

[31] Distributed European Infrastructure for Supercomputing Applications (DEISA), www.deisa.org

[32] Enabling Grid for E-sciencE (EGEE) www.eu-egee.org

[33] EUROGRID project, www.eurogrid.org

[34] European DataGrid project (EDG) www.eu-datagrid.org

[35] GLOBUS Project, www.globus.org

[36] Logistical Runtime System (LoRS), loci.cs.utk.edu/lors

[37] North Carolina BioGRID, www.ncbiogrid.org/

[38] Open Grid Service Architecture – Data Access and Integration (OGSA-DAI) www.ogsadai.org

[39] TeraGrid, www.teragrid.org

[40] Handle System, www.handle.net

[41] Siddiqui, M., Villazon, A., Hofer, J. and Fahringer, T. “GLARE: A Grid Activity Registration, Deployment and Provisioning Framework” Proceedings of Supercomputing 2005