CESNET Technical Report 2/2008

Multiple Ligand Trajectory Docking Study – Semiautomatic Analysis of Molecular Dynamics Simulations using EGEE gLite Services

Aleš Křenek, Martin Petřek, Jan Kmuníček, Jiří Filipovič, Zdeněk Šustr, František Dvořák, Jiří Sitera, Jiří Wiesner, Luděk Matyska

Received 26.2.2008

Abstract

Interactions between large biomolecules and smaller bio-active ligands are usually studied through a process called docking. Its aim is to find an energetically favorable orientation of a ligand within an active site of a biomolecule. Active sites are places where a chemical reactions take place and the role of the ligand is either to speed up, slow down or change the reaction (e.g., an enzyme catalyzed hydrolysis), which is why it can have huge pharmaceutical or other commercial impact. We present a tool allowing to effectively manage and control typical workflow of docking parametric study. Selected subsets of ligands and protein trajectory snapshots can be displayed in three different views and further analyzed. Finally, the application supports spawning and steering underlying computations running on the Grid.

Keywords: Achetylcholinesterase, Docking, Enzyme, Ligand, Molecular Dynamics, Job Provenance, Grid middleware, Grid infrastructure

1 Studied Problem

1.1 Docking Search

The docking search for biomolecular complex structure is done on snapshots taken from the molecular dynamics trajectory describing the dynamic behavior of the biomolecule. Each snapshot is a specific structure the biomolecule has at a specific time (a frozen structure). For each snapshot, we calculate the best position of the ligand (i.e., the orientation where whole system does have the lowest energy). This is repeated for each ligand we investigate and, in the end, we receive a matrix containing energies of snapshot/ligand interactions. The lowest energy shown in this matrix is the primary result of such a study, but the whole matrix is of importance as it provides insight into the dynamics of the interaction.

The creation of such a matrix is, from the computing standpoint, a very demanding task. Furthermore, the researchers need assistance managing its complexity (to be sure the whole matrix is computed and no element forgotten). A sophisticated job submission system coupled with an archive (a provenance) of jobs already run is a necessary prerequisite for such studies.

1.2 Specific Usecase

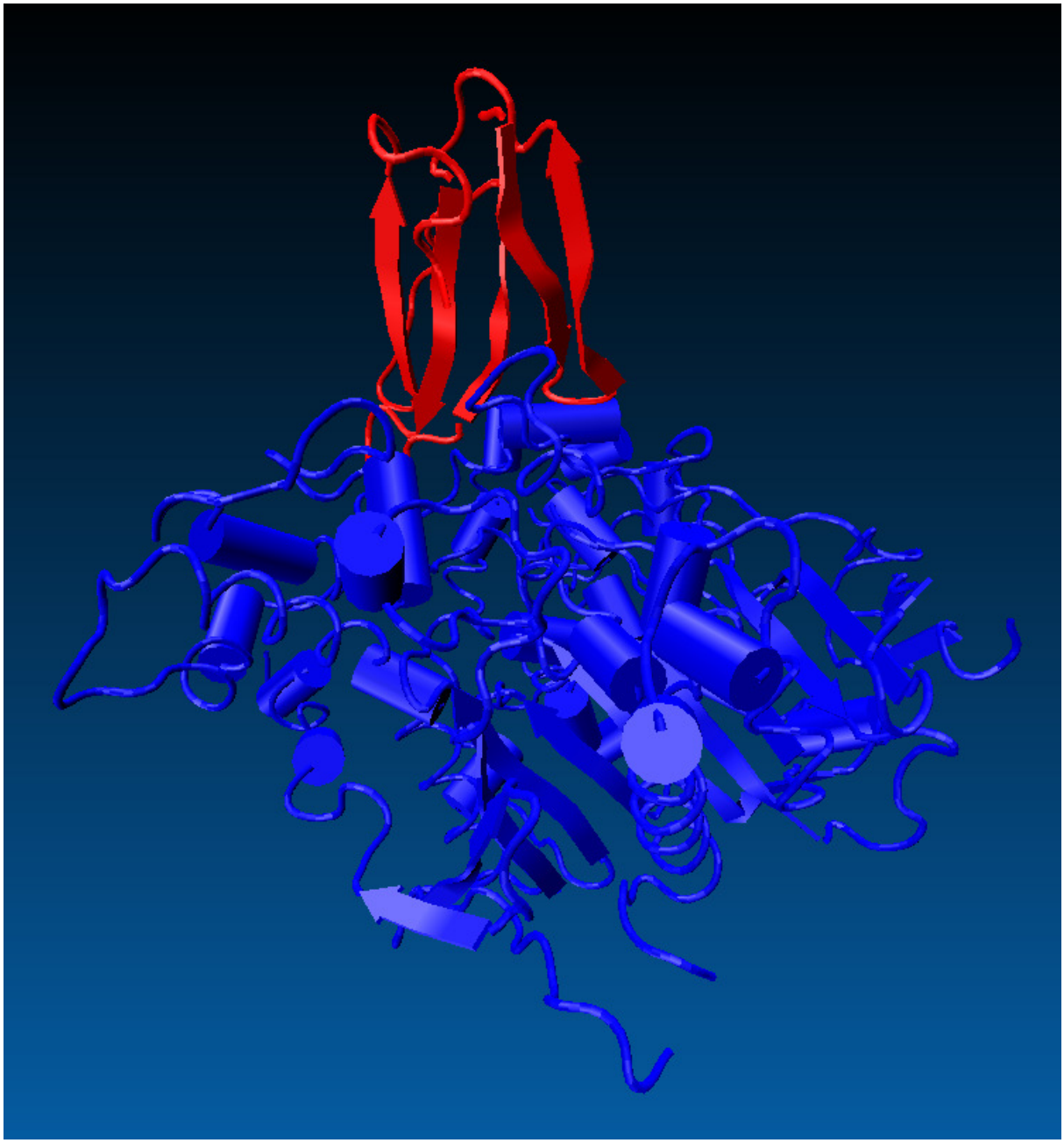

Figure 1. Three dimensional view of the secondary structure elements of human acetylcholinesterase as obtained from RCSB Protein database (code 1B41, colored by chains).

Nowadays, there are many organophosphate compounds, nerve paralytical substances (as sarin, soman, tabun, VX etc.) and pesticides, with ability to affect humans toxically. This effect is performed by irreversible inhibition of the acetylcholinesterase enzyme (AChE; EC 3.1.1.7) – serine hydrolase – that is crucial for the correct function of the human nerve signaling system (transmission of nerve signals across gaps between nerve cells). Acetylcholinesterase [1] is one of the most important enzymes in many living organisms, including humans and vertebrates,

and is found in the nervous system and in muscles.

AChE playes a key role in nerve signal transmission as its inhibition can lead to the very fast death of an organism (see Figure 1). AChE is responsible for regulating acetylcholine concentration during nerve signal transmission. In the central and peripheral nervous systems, nerve signals are usually transmitted by the acetylcholine neurotransmitter. The transmission is terminated by the cleavage of acetylcholine by AChE directly in the synaptic gap. A reactivation process whereby the catalytical potency of the inhibited enzyme can be restored, has been known for a long time. The reactivation procedure results in a free active enzyme that can play its physiological role again and a complex of AChE reactivator-inhibitor that is subsequently removed by the organism through detoxification. Unfortunately, there is no universal reactivator able to reactivate AChE enzyme inhibited by the most of commonly used nerve agents. Therefore, we attempt to study structural and energetical aspects of the reactivation process by means of computational chemistry to find suitable generic reactivators able to liberate AChE poisoned by nerve paralytic compounds.

In this study, we have investigated the interaction between acetylcholinesterase participating in nerve signal transmission, and a set of organic aromatic compounds that could serve as reactivators. We focuses on a detailed investigation of reactivators binding (through both weak van der Waals and electrostatic interactions) into the acetylcholinesterase active site. The actual problem shown here deals with a 2-ns acetylcholinesterase trajectory and 3 ligands, requiring approx. 6000 CPU hours on an average computing server. In reality, such studies use more and longer trajectories (tens of ns) as well as a higher number of potential ligands (tens to hundreds). Such a computation is unmanageable without semi-automatic support tools.

1.3 Application Software Used

The molecular dynamics trajectory of the acetylcholinesterase was calculated using molecular dynamics programs from the AMBER package, the docking procedure itself was carried out by DOCK[1]. VMD [2] was used to allow visual inspection of the resulting biomolecular complexes using our in-house visualization plugin.

2 Analysis Automation

The whole docking study is run from a custom graphical desktop application (or workbench, or dashboard ...) shown in Fig. Figure 2. It was designed under tight cooperation with NCBR[2] (National Centre for Biomolecular Research), researchers to suit their typical workflow for day-to-day analytical work, and – gradually – to take care of all common tasks performed, up to now, manually.

As the first step, the application supports the selection of the working domain for the current session, i.e., subsets of both trajectory snapshots and specific ligands. Then it queries the underlying Grid services and local job repository (Section 3.4.2),

[1] `http://dock.compbio.ucsf.edu/`

[2] `http://ncbr.chemi.muni.cz`

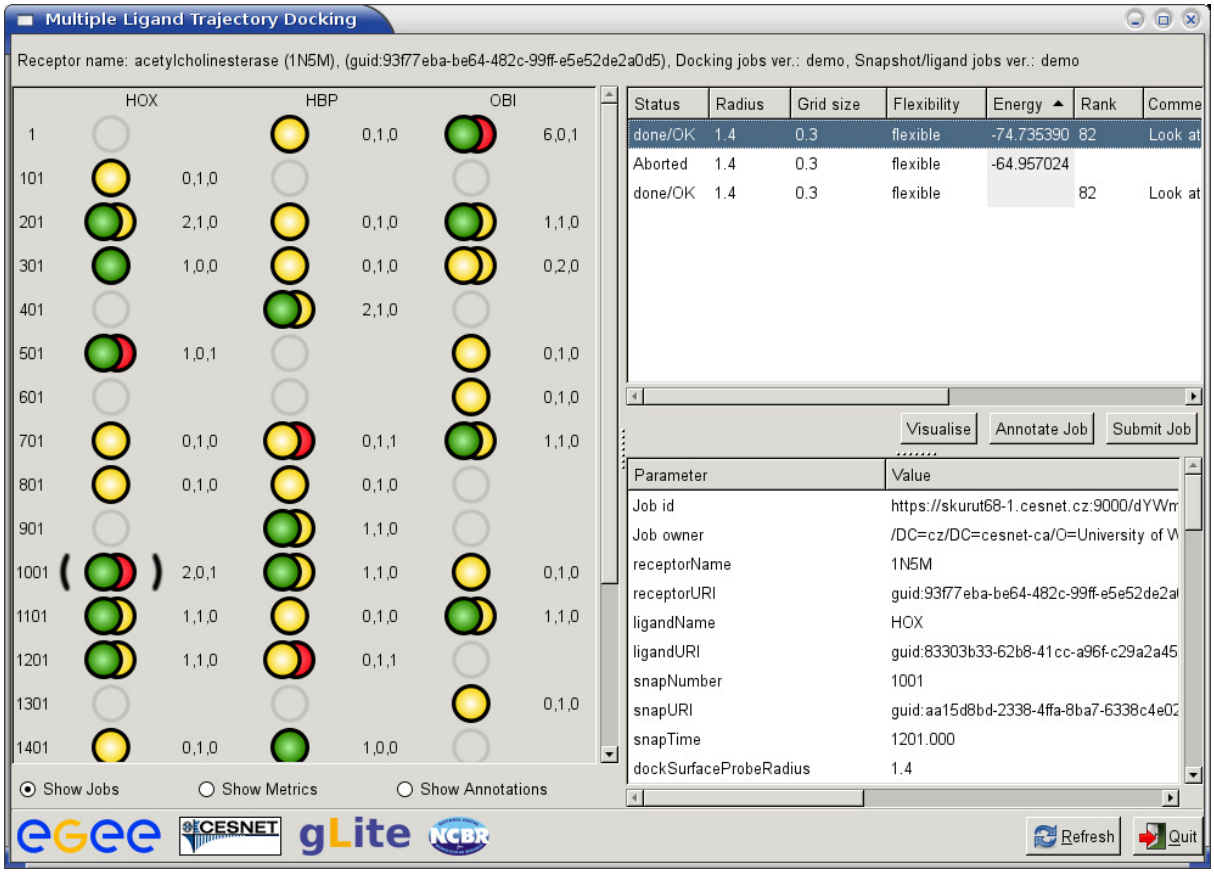

Figure 2. Basic layout of the graphical application. The left pane is a view on the snapshot vs. ligand array (both "middleware" and "application" views are shown), the top-right pane shows individual jobs falling into the selected array cell, and the bottom-right pane displays details of the selected job.

and displays a 2D array of Grid jobs (including prepared, running, and finished jobs) matching the criteria. Three viewing modes (left pane in Figure 2) of the array are provided:

— *Middleware oriented*: shows the number and status of Grid jobs falling to each array cell (green – finished successfully, red – failed, yellow – being processed).
— *Application metrics*: a significant numeric value (binding energy in this case) computed by each of the jobs is mapped to a color-scale representation, giving an immediate insight in its distribution over the working domain.
— *User rank*: similar to the previous one but assigned manually by users as a result of their expert assessment of the outcome of the computations.

The use of Grid services, rather than local and private user job repository, adds an important feature –*sharing the results and assessments* among all users of a collaborating team. Jobs (and their results) submitted by one user are immediately visible to all members of the team (obviously respecting security restrictions; the users have to set appropriate permissions). In this way, results are shared easily and duplicate computations can be avoided.

A typical user session starts with selecting the work domain (as described above). Then, results of finished jobs can be examined in detail, including three dimensional visualization of emerging complex 3D structures. Batches of jobs can be prepared

in order to fill empty cells of the array to complete overall docking analysis. Also, existing jobs can be used as templates, cloned, and re-run with modified input parameters.

Finally, the user can mark each job with a numeric rank (which is visualised with color scale in the "user rank" view described above) as well as a free-form text annotation recorded and visible to the others. For example, a job which achieves good (low) binding energy (the application metrics) can be annotated "good energy but due to apparently faulty computation" and assigned bad user ranking in order to exclude it from further considerations.

The description confirms that the application is designed specifically for solving this class of scientific problems. This was done intentionaly. In this case we do not believe in the "one size fits all" paradigm; requirements of different user communities can be rather diverse, and trying to design a generic application would yield an over-complex, cumbersome implementation. On the other hand, with the use of the Grid services in the background, the application is exteremely lightweight. After the design was agreed upon, the actual coding required only approx. two person-weeks of a skilled programmer.

3 Underlying Grid Services

3.1 Charon Extension Layer – Managing Job Submission

The Charon Extension Layer toolkit [3] is a universal framework creating a layer on top of the basic Grid middleware environment and making the access to the complex Grid infrastructure much easier compared to the native middleware. It provides a command-line oriented interface and is intended for users who require full control over their running computational jobs. CEL provides uniform and modular approach to complex computational job submission and management, and forms a generic system for the use of application programs in the Grid environment independently of Grid middleware present at the specific fabric infrastructure. CEL can be easily used for powerful application management enabling single/parallel execution of computational jobs without job script modification. Simultaneously, standard job management involving easy job submission, monitoring, and result retrieval can be performed without any additional hassle or requirements put on users.

The complete Charon Extension Layer combines two distinct subsystems – the Module System and the Charon System. the Module System is used to manage available application portfolio. It solves problems related to the execution of applications on machines with different hardware and/or operating systems, and it is also capable of simplifying the execution of applications in parallel environments. The Charon System is a specific application managed by the Module System that introduces a complete solution for job submission and subsequent management.

3.2 gLite Job Processing

The only way the user can access computational resources in gLite middleware[3] is through a *job*. Despite not completely restricted to, gLite is designed to support a large number of traditional batch, non-interactive jobs.

Upon creation, the job is assigned a unique immutable *job identifier* (jobid). The jobid is used to refer to the job all the time during the active life of the job and afterwards.

The user describes the job (executable, parameters, input files etc.) using the *Job Description Language (JDL)* [7] based on the extensible *Classified Advertisement* (ClassAd) [5] syntax. The description may grow fairly complex and include information on execution environment-related requirements, proximity of input and output storage, etc.

Job processing can be summarized as follows (denoting gLite components in italics):

- the job is submitted via the *User Interface* (simple command line tools)
- the *Workload Manager* (WM) [] queues the job and starts looking for a suitable *Computing Element* (CE) to execute it
- the job is passed to the chosen CE and runs there
- as a part of its processing, the job may download inputs from *Data Management services* [] as well as upload its main results there to be stored permanently
- after completion, the user can retrieve miscelaneous (volatile) job output directly
- all the time, the job is being tracked by the *Logging and Bookkeeping* (L&B) service [10], which provides the user with information on the job state and further details of job processing
- after the user retrieves the job output, the middleware data (namely the job trace in L&B) on the job are passed to *Job Provenance* (JP, Section 3.3) and purged from their original locations
- annotations can be added to the job during its active lifetime via L&B (even from the inside of running applications), or any time afterwards via JP.

3.3 Job Provenance – Archiving the Data

The need for a Grid middleware service that would help users track their jobs, store the information for a long term, allow adding further annotations, and, finally, provide efficient querying capabilities, was the primary motivation for developing the *Job Provenance* (JP) Service.

Pragmatic implementational requirements on JP, given its main purpose, are rather contradictory. Information on each job should be sufficiently detailed in order to allow job re-execution, while the data gathered should be stored for a long time. This implies ever growing storage space requirements that must be kept reasonable by making job records as compact as possible. The EGEE project aims at 1 million of jobs per day; quantitative assessments of implications in JP, as well as deployment considerations, are given in [10]. At the same time, efficient queries

[3] `http://glite.web.cern.ch/glite/`

are required, which is virtually impossible with such a huge number of compact records. Finally, JP has to be able to cope consistently with long-term evolution of various data formats.

The overall JP design tries to keep these requirements in a reasonable balance. This section provides an overview. Further details can be found in [6].

3.3.1 Data in JP and Their Organization

In JP, data are organized primarily on a per-job basis, a concept following the L&B model. Every data item stored in JP is associated with an actual Grid job. The following data are gathered from the Grid middleware:

— *job inputs*, directly required for job re-running: complete job description (the JDL record) as submitted to the WM system (WMS), and miscellaneous input files (gLite WMS input sandbox) provided by the user (however, job input files from reliable remote storage are not copied to JP; this would not be feasible in a large scale)

— *job execution trace*, documenting the job execution environment – complete L&B data, that is when and where the job was planned and executed, how many times and for what reasons was it resubmitted, etc. This also includes the results of "measurements" taken on computing elements, for example versions of installed software, environment settings, etc.

— *job annotations* – the JP service allows users to add arbitrary annotations to jobs in the form of "name = value" pairs. These annotations can be recorded either during job execution, or at any time afterwards. Besides providing information on the job (for example that it is a production-phase job of a particular experiment), these annotations may carry information on relationships between jobs and other entities such as external datasets, forming the desired data provenance record.

Figure 3 shows the data flow channels from Grid middleware components (gLite) into JP.

In order to overcome the diversity of various data formats as well as their long-term evolution, to provide further extensibility, and to unify the handling of different data, the JP concept distinguishes between the following views on the data:

— *Raw representation* – the physical data received and stored in JP. There are two input and storage modes in JP:

- Small size *tags*, expressed as "name = value" pairs, enter the system via its primary interface (a web service operation in the current implementation). "Value" is assumed to be a literal without any structure that JP should be aware of.
- *Bulk files*, typical example is the complete dump of L&B data or the job input sandbox, are uploaded via a suitable transfer protocol. Files are supposed to be structured. However, they are stored "as is", and upon upload they are annotated with format identification, including the version of the format. JP allows installing plugins that handle particular file formats (see below) understand the file structure, and extract required information.

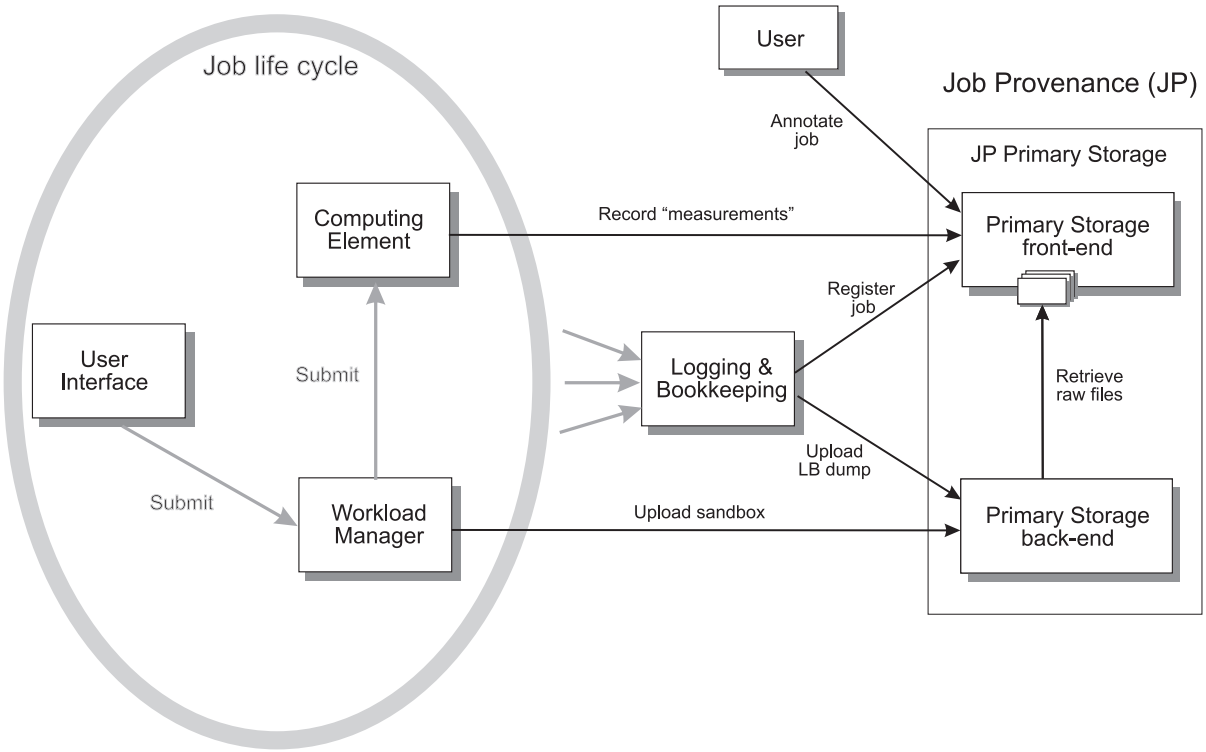

Figure 3. Data flow into gLite Job Provenance

— *Logical view* is an abstract level used to manipulate JP data, and it is the preferred way for the most of JP operations (queries are specified in terms of attributes of the logical view). The following list summarizes the basic ideas:
  - All data are expressed by *attributes* at the logical level. A job attribute has a unique name and may have multiple values for a single job. The attribute name must be fully qualified with a namespace (its schema can be specified and enforced) in order to ensure extensibility and uniqueness.
  - Explicitly recorded tags (user annotations) map to attributes in a straightforward way, name and value of a tag becoming name and value of an attribute.
  - An uploaded file is usually a source of multiple attributes, which are "digested" from the file based on knowledge of a particular file structure and semantics. For example, the L&B dump file provides attributes such as job submission and completion time, number of resubmits, CE where the job ran, etc.
  - The "digest" process extracting attributes from raw data files is implemented through *JP plugins*. The task of a plugin is to parse a particular file type and provide calls to retrieve attribute values. JP defines a fixed plugin API.

3.3.2 JP Components

JP provides two classes of services: a permanent *Primary Storage* (JPPS) accepts and stores job data, while the possibly volatile and configurable *Index Servers* (JPIS) provide an optimized querying and data-mining interface for the end-users (see Figure 4). The relationship between JPPS and JPIS ranks as many-to-many – a single

JPIS can query multiple JPPS's and vice versa, a single JPPS is ready to feed multiple JPISs.

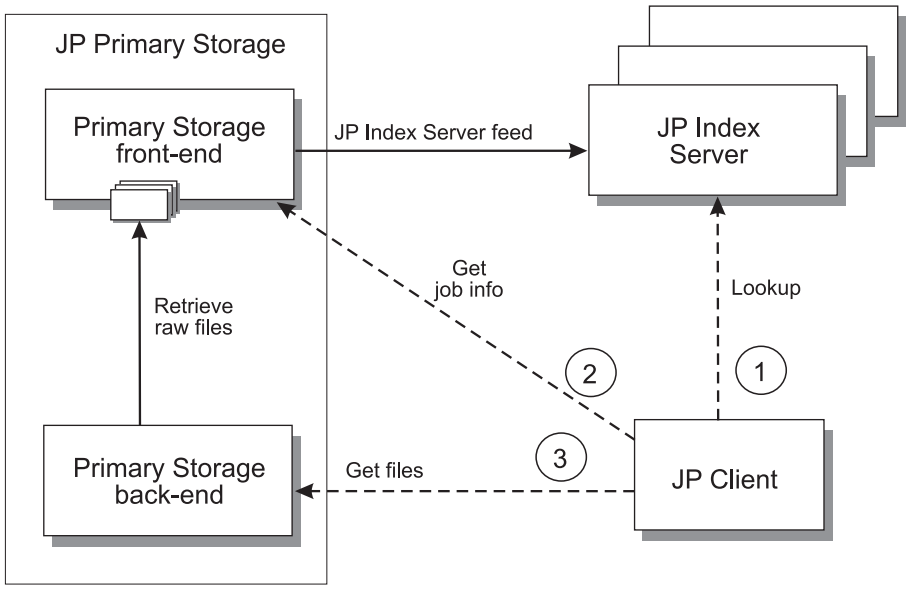

Figure 4. Job Provenance components

Primary Storage

A single instance of JPPS, shown in Figure 4, is formed by a front-end exposing its operations via a web-service interface [8], and a back-end responsible for actual data storage and providing the bulk file transfer interface using arbirtrary file-oriented transfer protocol(s). Both the front- and back-end share a filesystem so that the file-type plugins linked into the front-end access their files via POSIX I/O.

JPPS operations fall into the following categories:

— *Job registration.* Each job has to be explicitly registered with JP. Currently the registration is done transparently by the L&B server upon job submission (in parallel with the job registration in L&B, though not blocking the job submission).
— *Tag recording.* Add user tags (annotations) in the "name = value" form to JP job records.
— *Bulk file upload.* File properties (type, optional name etc.) are specified via the front-end interface, the upload itself goes directly to the back-end (using `gridftp` transfer).
— *Index Server feed* allows JPIS to ask for batch feed as well as register for incremental updates.
— *Data retrieval.* The only direct data retrieval supported by JPPS is keyed by the jobid. Both individual attributes and the whole files can be retrieved.

Primary Storage covers the first set of requirements specified for the Job Provenance – storing compact job records, allowing users to append annotations, and providing elementary access to the data.

The current implementation uses the MySQL relational database to store basic job metadata and a Globus `gridftp` server as the back-end.

Index Server

The role of *Index Servers* (JPIS) is to process and re-arrange the data from Primary Storage(s) into a form suitable for frequent and complex user queries. A typical interaction is shown in Figure 4, consisting of following steps:

1. The user queries one or more JPISs, receiving a list of IDs of jobs matching the query.
2. JPPS is directly queried for additional job attributes or URLs of stored files.
3. The required files are retrieved.

The querying language is intentionally restricted in order to allow efficient implementation of the query engine. The current format of the query is a list of lists of conditions. A condition is a comparison (less, greater, equal) of an attribute value to a constant. Items of an inner list must refer to the same attribute and they are logically OR-ed. Finally the inner lists are logically AND-ed. According to our experience with the L&B service, this query language is powerful enough to satisfy user needs while simple enough to allow efficient processing.

For example, to query all jobs named "dock", executed with a "flexible" parameter, and ran on Monday or Tuesday:

```
(program="dock") AND
(param="flexible") AND
(day="Monday" OR day="Thuesday")
```

JPIS provides the following operations:

— *User queries* – it is the primary interface for end users as described above. A query specifies a set of attributes to be retrieved and conditions determining the set of matching jobs.
— *JPIS-JPPS communication* – implements the data flow from JPPS to JPIS. Each JPIS can subscribe to JPPS according to its configuration to receive data matching the configuration or just ask for data matching the configured criteria.
— *Administrative* – calls to change JPIS configuration without interfering with its normal operation.

Index Servers are created, configured, and populated semi-dynamically according to the needs of a particular user community. The configuration consists of:

— one or more Primary Storages to contact,
— conditions (expressed in terms of JP attributes) on jobs that should be retrieved,
— list of attributes to be retrieved,
— list of attributes to be indexed – a user query must refer to at least one of these for performance reasons.

The set of attributes and the conditions specify the set of data that is to be retrieved from JPPS, and they reflect the assumed pattern of user queries. The amount of data fed into a single JPIS instance is assumed to be only a fraction of data in JPPS, both regarding the number of jobs, and the number of distinct attributes.

The current JPIS implementation keeps the data also in a MySQL database. Its schema is flexible, reflecting the server configuration (columns are created to hold particular attribute value, as well as indices). Currently all attributes are handled

as strings, however, we are considering type-extension mechanisms that would allow processing complex attribute types, as well as adding further operators besides simple comparisons.

3.4 Application specific configuration and issues

3.4.1 Job Types

The analysis described in Section 2 requires running computational jobs of three classes:

— *Snapshot preparation.* The job extracts a given snapshot from the Molecular Dynamics (MD) trajectory, stores the data file in permanent storage, and records information on the snapshot (namely the trajectory name, snapshot number, data file location at the permanent storage, and several characteristic numbers such as its internal energy) into JP.

— *Ligand preparation.* The job performs ligand optimization, stores datafile in permanent storage, and records information on the ligand (its name, data file location, and several characteristics) in JP.

— *Docking.* These jobs do the actual docking computation. First, information on the job inputs is recorded into JP in order to make the job visible in the array immediately after submission. Then, required input files are downloaded, and the `dock` executable is invoked using specified parameters. Finally the results of the docking computation are uploaded, and metadata describing the result (data files location again, and characteristics of the solutions) are stored in JP.

Appendix A provides a complete list of JP attributes attached to these job types.

The first two classes of jobs are expected to be run once in a batch, as an initial step of the whole analysis. They determine a possible working domain. However, the domain can be extended later with adding more snapshots and/or ligands (by runnig more of these preparatory jobs).

On the contrary, the docking jobs are run routinely during the analysis. Those are the jobs that are submitted by the graphical front-end and managed in the local job repository.

3.4.2 Local Job Repository

Grid services exhibit certain intrinsic but rather user-unfriendly properties, namely asynchronous behaviour, unexpected failures, and slow response to operations. Therefore, it is desirable to shield the user from this behaviour with suitable wrapping of the services. On the other hand, it makes little sense to provide such wrapping with an additional *service*; the same problem would occur, just in a different location. On the contrary, we address the issue locally, with data stored and auxiliary programs running solely on the machine where the user graphical front-end is run.

The complexity and eventual failures of job submission and job output retrieval are handled by the Charon system (Section 3.1). The system uses a per-job dedicated working directory where various job metadata are stored. We extend this

approach by adding further metadata files that control interaction with JP. Within this metadata storage, a job is handled in the following steps:

1. On job submission, the graphical user interface calls a library function that creates a dedicated job directory, stores all job parameters there, and creates initial metadata. The semantics of the library call is completely local, it returns immediately, not being affected by eventual unavailability of Grid services.
2. A local daemon starts checking the directory periodically. The job is submitted (via Charon).
3. After successful submission, job JP data which are known at that time (e.g., snapshot number and ligand name) are stored into JP to be available for queries immediately.
4. After successful job completition, its miscelaneous output is retrieved, and the job protocol(see Appendix C) is uploaded to JP.

Steps 3 and 4 are prototype implementations only. In the planned full JP integration, the job input data for JP (step 3) will be included in the job description and stored in JP automatically by the middleware services. Similarly, the job description will denote the protocol file name and it will be uploaded to JP as a part of the job clean-up procedure.

Besides handling the job processing, the local repository serves as a backup backend for queries called by the graphical front-end. Therefore, the user can see job-related data immediately after its submission through the GUI (even if the job has not been submitted to the Grid yet). This also means that the operation of the GUI is not critically affected by the unavailability of Grid services (information on other users' jobs is not visible in such case).

3.4.3 Service Configurations

Besides standard configuration of gLite services, the following specific items have to be addressed:

— *L&B interface to JP*: The L&B server used for the analysis must be configured to propagate job registrations and upload finished job data to JP Primary Storage. (This is supported by standard configuration but not enabled by default).
— *JP Primary Storage plugin*: the jobs upload docking protocol to JP. It may be used to extract many of the job attributes. A specific plugin[4] performing this extraction must be installed.
— *JP index server configuration* must support all the queries of the graphical front-end. Details are given in Appendix B.

3.5 Run Infrastructure

The complete set of application services together with the graphical interface were developed and implemented in the subset of EGEE grid infrastructure – Virtual Organization for Central Europe (VOCE). VOCE [9] is a dynamic, multi-institutional community established by resource providers within the Central Europe region (the

[4] `http://lindir.ics.muni.cz/dg_public/cvsweb.cgi/uf07/dock_plugin`

CE Federation in the EGEE terms). It directly supports CE researchers by providing the storage and computing services. VOCE uses the gLite Grid middleware as provided by EGEE to support data sharing and computational resources within the CE. It also provides a platform on which other Grid and application software can be installed and used to solve various types of computational or data-intensive jobs.

Unlike majority of other virtual organizations, VOCE tends to be a generic virtual organization (VO) providing application-neutral environment especially suitable for Grid newcomers allowing them to quickly acquire initial Grid computing experience and to test and evaluate Grid environment towards their specific application needs. VOCE environment is also suitable for small groups for whom creating and maintaining their own VOs may represent too much overhead.

The primary goal of VOCE is to provide an environment where new and "small" end user groups can use a production level Grid, adapt existing or develop new applications, and prepare themselves to eventually start their own VOs. Its secondary goal is to provide an environment where new middleware services can be introduced in a fast way, including services developed by the EGEE CE partners independently of the main-stream gLite development.

4 Conclusion

Scientific experiments, even those carried in a pure computational environment, require very precise recording of the experiments, both of their setup and results. This is even more critical for parametric studies, where similar experiments/computations are carried on a large multi-dimensional domain of possible inputs.

We present a semi-automated approach to managing such records using available Grid services of the gLite middleware and the Charon Extension Layer toolkit. Besides keeping the records for an individual user, the approach also strongly supports collaboration in a user group. On the other hand, the user is shielded from the complexity of the Grid where desirable.

To demonstrate the feasibility of these ideas, we have developed a specialized graphical interface for solving generic biomolecular parametric jobs. It allows user application metrics evaluation based on targeted parameters with potential extension for extensive biomedical screening. Following features are available as standard services: computational jobs manipulation (input modification, jobs resubmission), targeted search and selection of desired jobs (finished, non-finished, aborted). Moreover, the whole application is based od modular approach allowing incorporation of application-specific plugins for the presentation of results (e.g. visualization).

Currently, the prototype is being used by users at the National Centre for Biomolecular Research (NCBR), and it is being further extended according to their requests. We have also successfully demonstrated its use during the EGEE User Forum conference[5].

This study demonstrates the usability of the gLite software stack to deal with complex computational studies in the computational chemistry area with the po-

[5] `http://egee.cesnet.cz/mediawiki/index.php/Job_Provenance_Demo`

tential for many others application domains.

5 Acknowledgements

This work was done within EU EGEE-II project (INFSO-RI-031688), and supported by the Ministry of Education of the Czech Republic (contract no. MSM0021622413), and the Grant Agency of the Czech Republic (contract no. 204/03/H016). Special thanks to prof. Jaroslav Koča for allowing start of CEL development and all subsequent support.

References

[1] WIESNER, J.; KŘÍŽ, Z.; KUČA, K.; JUN, D.; KOČA, J. Computer Modeling and Simulation – New Technologies in Development of Means against Combat Chemical Substances. *Voj. zdrav. listy*. 2005, vol. 46, no. 2, p. 1144–1145.

[2] HUMPHREY, W.; DALKE, A.; SCHULTEN, K. VMD – Visual Molecular Dynamics. *Journal of Molecular Graphics*. 1996, vol. 14, p. 33-38.

[3] KMUNÍČEK, J.; KULHÁNEK, P.; PETŘEK, M. Charon system – framework for applications and jobs management in Grid environment. In *Proceedings of Cracow Grid Workshop 2005*. Academic Computer Center CYFRONET AGH, 2006, p. 332–340. ISBN 83-915141-5-3.

[4] KMUNÍČEK, J.; PETŘEK, M.; KULHÁNEK, P. Charon extension layer – universal toolkit for Grid applications and computational jobs maintenance. In *Proceedings of Cracow Grid Workshop 2006*. Academic Computer Center CYFRONET AGH, 2007.

[5] RAMAN, R.; LIVNY, M.; SOLOMON, M. Matchmaking: distributed resource management for high throughput computing. In *Proceedings of HPDC*, 1998.

[6] DVOŘÁK F. et al. gLite job provenance. In *Proceedings of IPAW'06*. Springer, 2006, LNCS 4145, p. 246–253.

[7] PACINI F. et al. *Job Description Language Attributes Specification*. CERN EDMS #590869, 2006.

[8] EGEE JRA1 team. *EGEE Middleware Design—Release 1* CERN EDMS #567624, 2005.

[9] KMUNÍČEK, J.; KOUŘIL D. Central European Grid infrastructure for generic applications. In: *Proceedings of 2nd Austrian Grid Symposium*. Ossterreichische Computer Gesellschaft, 2007, p. 131–142.

[10] MATYSKA, L. et al. *Job Tracking on a Grid — the Logging and Bookkeeping and Job Provenance Services*. Technical report 9/2007[6], Praha: CESNET, 2007.

[6] http://www.cesnet.cz/doc/techzpravy/2007/grid-job-tracking/

Appendix A. Job Attributes

The following tables shows a complete list of job attributes stored in JP.

A.1 Snapshot preparation jobs

jobClass
: class of job (snapshot), see Section 3.4.1

version
: identification of the experiment (all jobs in analysis share this id, others are excluded)

snapURI
: where the snapshot is stored

snapNumber
: snapshot number

receptorName
: receptor name

snapTime
: snapshot time on the trajectory

receptorPDB
: receptor name according to Protein Data Bank[7]

snapTemp
: temperature of the snapshot

snapRMSD
: snapshot RMSD (geometrical difference from reference shape)

snapEPot
: electrostatic potential of the snapshot

A.1 Ligand preparation jobs

jobClass
: class of job (ligand), see Section 3.4.1

version
: identification of the experiment (all jobs in analysis share this id, others are excluded)

ligandName
: short ligand name

[7] `http://www.pdb.org`

IUPACName
: full ligand name

SMILES
: ligand formula

sourceDB
: from which database the ligand is taken

qmOptMethod
: ligand optimization method

qmOptBasis
: wave function basis used for ligand optimization

qmEspMethod
: method to compute ligand electrostatic charges

qmEspBasis
: basis used to computed charges

atomTypes
: atom types for the force field used in computation

fileFormat
: format of the ligand file

ligandURI
: where the ligand is stored

A.1 Docking jobs

jobClass
: class of job (docking), see Section 3.4.1

version
: identification of the experiment (all jobs in analysis share this id, others are excluded)

snapURI
: where the snapshot is stored

snapNumber
: snapshot number

receptorName
: receptor name

snapTime
: snapshot time on the trajectory

receptorURI
: auxiliary template mol2 file for the docking job

ligandURI
: where the ligand is stored

dockSurfaceProbeRadius
: radius of probe for the docking calculation

dockResidueList
: residues determining the active site where docking positions are searched

dockDistance
: maximum distance from the residues to search for docking positions

dockGridRes
: resolution of the calculation grid

dockNumScoredConf
: number of docked conformers to search (flexible docking only)

dockMethod
: flexible or rigid docking

dockProtocol
: protocol file name

gridMin{X,Y,Z}
: minimal coordinates of the searched grid

gridMax{X,Y,Z}
: maximal coordinates of the searched grid

gridScore
: value of the scoring function

conformerVdwEnergy
: van der Waals energy of the resulting conformer

conformerESEnergy
: electrostatic energy of the resulting conformer

structureURI
: where the resulting structure was stored

userComment
: user-assigned free-form annotation of the docking result

userRank
: user-assigned numeric ranking of the docking result

Appendix B. Index server configuration

JP Index server configuration (see Section 3.3.2) includes JP Primary storage (s) to query, attributes to retrieve from JPPS, conditions specifying which jobs should be retrieved, and attributes to be indexed. We use the following values for our experiment:

— *JPPS query conditions*: for the prototype testing users involved are enumerated (i.e., records on all their jobs are retrieved). Production configuration would be based on a specific value of the version attribute (identification of the experiment), though.

— *Retrieved attributes*: all attributes listed in Appendix A and the following JP/L&B system ones:

Attribute	Description
owner	job owner
jobId	grid job id
regtime	job registration time
RB	resource broker handling the job
CE	computing element where the job ran
UIHost	worker node of the CE
CPUTime	consumed CPU time
finalStatus	final job status
finalStatusDate	when the job reached the final status
retryCount	number of retries

— *Indexed attributes*: owner, jobId, CE, finalStatus, jobClass, version, snapTime, snapGeom, ligandURI, snapURI, receptorURI, receptorName

The complete configuration is available online[8].

Appendix C. Docking Protocol

An example of docking protocol of a real job is given bellow. This XML file is uploaded "as is" into JP and parsed there by the plugin to produce several job attributes (Appendix A). Note multiple occurences of `<conformer>` element which yield multi-valued JP attributes.

```
<?xml version="1.0"?>
<dockOutput xmlns="http://egee.cesnet.cz/en/Schema/JP/Docking">
  <proteinSurface unit="angstrom2">24776.54</proteinSurface>
  <grid unit="angstrom">
    <gridMinX>38.937000</gridMinX>
    <gridMinY>41.829000</gridMinY>
    <gridMinZ>37.361000</gridMinZ>
    <gridMaxX>70.847000</gridMaxX>
    <gridMaxY>68.885000</gridMaxY>
```

[8] http://lindir.ics.muni.cz/dg_public/cvsweb.cgi/uf07/JPIS/glite-jpis-config.xml

```
    <gridMaxZ>78.851000</gridMaxZ>
  </grid>
  <conformers
    conformersFound="3"
    conformersURI="guid:6084d8b5-73da-4460-9874-5a921611e952"
  >
    <conformer id="1">
      <gridScore>-74.735390</gridScore>
      <conformerVdwEnergy>-55.391171</conformerVdwEnergy>
      <conformerESEnergy>-19.344217</conformerESEnergy>
    <structureURI>guid:6ba7e1cd-52fc-4977-b538-affd06f6322f</structureURI>
    </conformer>
    <conformer id="2">
      <gridScore>-74.723007</gridScore>
      <conformerVdwEnergy>-55.614491</conformerVdwEnergy>
      <conformerESEnergy>-19.108521</conformerESEnergy>
    <structureURI>guid:a59d6c33-caf6-4d6a-8b4c-e4eba05628b6</structureURI>
    </conformer>
    <conformer id="3">
      <gridScore>-74.651627</gridScore>
      <conformerVdwEnergy>-55.645340</conformerVdwEnergy>
      <conformerESEnergy>-19.006287</conformerESEnergy>
    <structureURI>guid:91d9556e-d046-4504-be2f-6ff436881906</structureURI>
    </conformer>
  </conformers>
</dockOutput>
```