Efficient Data Storage and Analysis for Generic Biomolecular Simulation Data

Muan Hong Ng[1], Steven Johnston[1], Stuart Murdock[2], Bing Wu[3], Kaihsu Tai[4], Hans Fangohr[1], Simon Cox[1], Jonathan W. Essex[2], Mark Sansom[4] and Paul Jeffreys[3]
[1]Southampton e-Science Centre and [2]School of Chemistry, University of Southampton, UK
[3]Oxford e-Science Centre and [4]Department of Biochemistry, University of Oxford, UK
correspondent email: mark@biop.ox.ac.uk

Abstract. BioSimGrid aims to deliver a biomolecular simulation data repository to enable more efficient storage, access and exchange of biomolecular simulation data. This is an ongoing project still in development that seeks to exploit the concept of the Grid where large computational and data resources are made available to users in a highly accessible manner. The ability to submit, search, query, retrieve, and post process biomolecular data in a uniform way is vital for the biochemistry community for more efficient data sharing. The study of biomolecular simulation data, which encapsulates the motions of molecules are key contributions to applications such as drug discovery. This paper describes the BioSimGrid project and relates it to other recent work on grid-enabled data storage. We describe the middleware that enables the work flow of BioSimGrid and finally discuss the future work.

1 Biomolecular Simulation and BioSimGrid

Computer simulations play a vital role in biochemical research. By simulating the interactions of all atoms within a molecule or protein, the biochemical properties of the structure can be revealed. One important application of such Molecular Dynamics and Monte Carlo simulations is predictive modelling in drug discovery, where the motion of proteins is important. These simulations are computationally demanding and they produce huge amounts of data (up to 10 GB each) which is analysed by a variety of methods in order to obtain biochemical properties. Generally, these data are stored at the laboratory where they have been computed in a proprietary format which is unique to the simulation code that has been used. This constrains the sharing of data and results within the biochemistry community: (i) the different simulation results are usually not available to other groups and (ii) even if they are exchanged, for example via FTP, then the data can generally not be compared easily with post-processing tools due to the varying data formats. This missing ability of sharing and comparing the simulation data is regarded as a major impediment to the discovery of new science in the biochemical community.

BioSimGrid [1] seeks to tackle this problem by enabling biochemists to deposit their simulation data of varying formats to a shared repository. This will allow biochemists to retrieve a slice or part of a protein in a uniform way for post-simulation analysis. By providing a uniform data storage and data retrieval mechanism, different proteins can be compared easily.

Figure 1 demonstrates typical scenarios of using the BioSimGrid project. The completion of a biomolecular simulation delivers simulation data, which is called a "trajectory". A trajectory consists of many frames (corresponding to time steps in the simulation process) of simulation data recording the positions and velocities of all atoms. The first step is to submit the new trajectory and all the relevant meta-data (which describes the simulation and will allow sophisticated querying of all submitted trajectories) to the database. The extraction of the meta-data (from the simulation configuration files) and the trajectories (from the simulation data files) is fully automated, but the user has the option to provide additional information, such as publication references which cannot be extracted from the simulation configuration and data files.

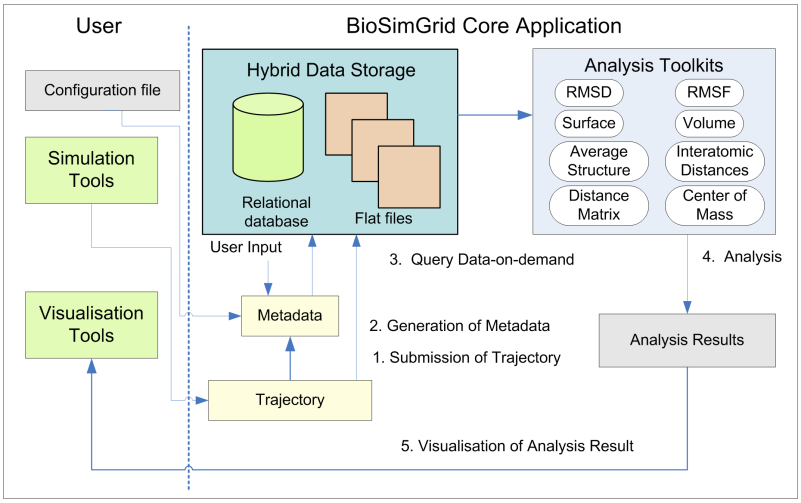

Fig. 1: A schematic sketch of the work flow in the BioSimGrid project.

Once the data is stored in the database, users can query different slices of one or more trajectories and perform a number of standard analysis computations (a selection is shown in the figure) on these data. The work flow is then completed by the graphical display of the analysis results (either as vector graphics, bitmaps, movies or using interactive 3-dimensional environments such as Visual Molecular Dynamics [2] and PyMol [3]). The results are, of course, also available in text or data files.

The following section of the paper discusses two related work on grid-enabled data storage. Section 3 describes the architecture of BioSimGrid where the data storage layer, the middleware layer and the user interface layer are discussed in details. Section 4 gives a brief outline on the current issues and the future work on the next prototype, and we finally conclude in section 5.

2 Related Work

2.1 GridPP and the European DataGrid Project

GridPP [4] is a collaboration project between Particle Physicists and Computer Scientist from the UK and Cern aiming to build a Grid for Particle Physics. One of the key components of GridPP is the European DataGrid Project (EDG) [5] which deals with managing a large quantity of sharable data reliably, efficiently and scalably. EDG aims at enabling access to geographically distributed computing and storage facilities. It provides resources to process huge amounts of data from three disciplines: High Energy physics, Biology and Earth Observation. EDG has a file replication service to optimise data access by storing multiple copies of local data at several locations. This replication framework has an optimisation component to minimise file access by pointing access request to appropriate replicas and proactively replicating frequently used files based on access statistics.

As compared to DataGrid, BioSimGrid aims to provide a mechanism of data access at a finer granularity level, by delivering a slice of a trajectory rather than a whole file. Hence the concept of file replication of DataGrid can potentially be adopted and modified to suit a finer granularity level of data access.

2.2 OGSA-DAI

OGSA-DAI (Open Grid Services Architecture - Data Access and Integration) [6] is a grid middleware that has been developed to allow access and integration of heterogeneous data sources as though they were a single, logical data resource. It is well suited particularly for database applications that involve legacy databases of different vendors. Its services offer data federation and distributed query processing to allow joint table query from multiple disparate data

sources. To enable integration of heterogenous databases, OGSA-DAI data query service introduces an extra middle layer on top of the data sources before data is brought into the processing environment. This potentially introduces a layer of overhead in the middleware. OGSA-DAI is an OGSA-referenced implementation of grid services. With the deprecation of OGSI which is the architectural implementation of OGSA, OGSA-DAI is currently migrating to web services standard space [7]. Hence to date, it is still an ongoing project which is yet to prove its maturity. One of the reasons OGSA-DAI is not employed in BioSimGrid is that the heterogeneity of data sources is minimal in this project, as we do not deal with legacy databases. Furthermore, BioSimGrid has multiple databases storing identical metadata (please refer to section 3.1), hence the functionality of distributed querying is not immediately required.

3 The Architecture of BioSimGrid

BioSimGrid seeks to fulfil the following criteria in its implementation:

- to minimise data storage, in order to store as many trajectories as possible in a fixed amount of storage space.
- to maximise data transfer rate, in terms of the speed of delivering data to the computational elements, in this case the post processing tools.
- to provide an abstraction of the data layer, where biochemists are freed from the complication of using and understanding data quering languages and the data storage structure in their scientific research.
- to provide a transparency of data location to the users, where actual physical location of the data is hidden.

As shown in figure 2, the architecture of BioSimGrid encompasses three layers: the data storage layer, the middleware and the user interface layer. Each of these will be described in the following sections.

3.1 BioSimGrid Data Storage Layer

The data storage layer is responsible for managing the data on a single machine and exposes methods which are used by the Data Retrieval component to provide the user with data. This layer is required on each machine that is storing trajectory data, initially there will be six remote sites each running this layer. It provides an API which abstracts from the method used to store the data and provides simple access methods for both querying and retrieving data. The trajectory data is divided into two key sections, the metadata and the coordinate data.

Trajectory Metadata

The metadata is additional information about the trajectory which can either be supplied by the user, the input files or calculated at a later stage. It also includes the topology which describes the structure of the protein (chains and residues). This metadata is comparatively small and can be replicated across all sites using standard database replication tools. The advantage of replicating the metadata across all sites is so that a user can query all the trajectories stored in the system by querying and single machine and expect a timely response. This design also helps with scalability and load balancing: since the volume of metadata is small, additional nodes can be added to the system and easily incorporated by simply replicating the database. Since each node stores the topology of all the trajectories, users can use any node to query and process data helping to balance the load across the system.

Trajectory Coordinate Data

The coordinates for every atom for every timestep are stored resulting in a large volume of data which has to be managed. We have devised a fast, efficient way to store the coordinates using flatfiles which reduces the storage requirements as well as improving performance results. This flatfile method was implemented using Python pickle [8] and it was compared with a commercial database (DB2) as well as an existing flatfile method (NetCDF [9]). The performance results are shown in Table 1. These results show that a flatfile method is well suited to our application for both random and serial data access.

We selected our own method for flexibility as a whole trajectory is broken into a set

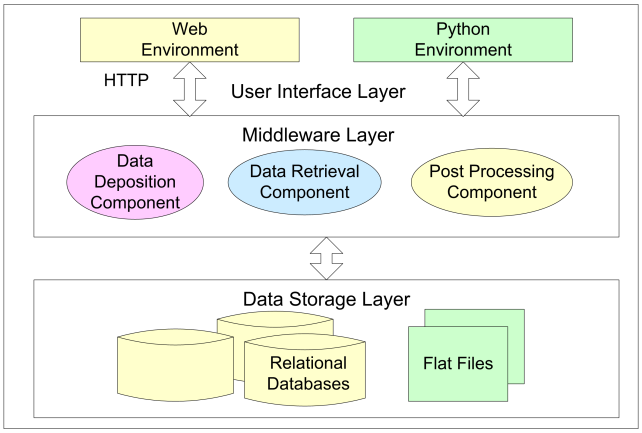

Fig. 2: The architecture of BioSimGrid depicting the data storage layer, middleware layer and user interface layer

of files which are then replicated to at least one other node. This helps to load balance the coordinate data requests as well as provides offsite backups of the data. This abstraction layer also permits the use of different storage methods which can include compression and custom formats, which are completely transparent to the user.

Currently only the coordinates are stored using this method but the next version will store both coordinates as well a velocities, using the same method.

	DB2	netCDF	Python pickle
Size(GB)	7.5	3.0	3.0
Random Access (Sec)	560.8	16.4	18.6
Sequential Access (Sec)	389.0	4.9	5.5

Table 1: Summary of performance results comparing different flatfile methods with a commercial database (DB2)

3.2 BioSimGrid Middleware

The middleware of BioSimGrid is implemented on a modular architecture to enable easy extension and future plug in. It is written in Python [8], a free, open-source and platform-independent high-level object-oriented programming language. Python is chosen for several reasons: (i) the biomolecular simulation community are moving towards Python as the preferred environment for post-processing analysis and several mature post-processing tools written in Python exist already (for example, MMTK [10] and PyMOL [3]). (ii) Python can easily integrate and interface to compiled codes so that other existing tools (typically written in Fortran or C) can be re-used immediately. (iii) Python comes with a substantial set of standard libraries which can be used in this project and avoid re-coding common tasks.

Data Deposition Component

The process of depositing a trajectory into the BioSimGrid database is completely automated and the complication of the underlying storage structure is abstracted from the users. One of the challenges is to cater for different simulation packages that produce simulation data in various file formats. To deal with this, the deposition component consists of different parsers for different simulation packages to parse the simulation data files into a generic input object. This object is then parsed through a validator to check for correct data type and their validity against various dictionaries (e.g. the existence of a residue in the dictionary). The process is completed when the validated generic input object is deposited into the flat files (coordinates and velocities) and database (metadata) through an importer. With the modular approach as shown in

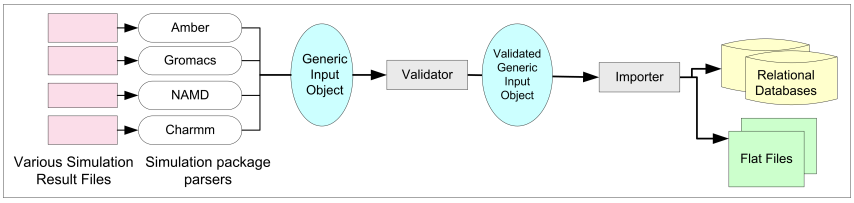

Fig. 3: The modular implementation of a data deposition component which includes a set of parsers, a validater and an importer. New parsers can be easily added to this modularised component.

```
from Deposit.NAMDDeposit import NAMDDeposit
files ={'coordinates': '/path/coordFiles',
          'topology':'/path/topoFile',
         'parameter':'/path/paraFile'}
g = NAMDDeposit(files)
```

Fig. 4: An example of a user script to deposit a NAMD trajectory. The underlying complexity of parsing, validating and importing of trajectory into the database is hidden from the users

figure 3, new parsers can easily be added for any new simulation package if required. The underlying complexity of parsing, validating and importing of trajectory into the database is hidden from the users. A biochemist needs only to run 3 lines of code to deposit their trajectories by specifying the path for their simulation data files, as shown in figure 4.

For the next prototype, the data deposition component will be extended to cater for the distributed nature of the application. We envisage an implementation of multiple deposition points to avoid single point failure and performance bottlenecks. In this case, a global identifier will be assigned to uniquely identify a trajectory and facilitate the synchronisation of multiple metadata databases. To deposit a trajectory from a remote location the generic input object will be serialised at the deposition client and deserialised at the deposition server.

Data Retrieval Component

The data retrieval component provides a single point of entry for all the trajectories stored on any of the sites. Each site will be running a data retrieval component and a user can use any site to query the data in the entire system. This component queries the local database to retrieve any metadata that is requested, so the user can query information about a trajectory on a different site without having the overhead of contacting the hosting site. This component abstracts the location of the trajectory data from the user and is responsible for getting coordinates from external sites if they are not stored locally.

Figure 5 shows how the data is transparently retrieved from a remote site so that it can be used by a users script. In step 1 and 2 the user submits a script that requests request for a set of coordinates from the Data Retrieval Component. This component first looks at the metadata database to retrieve the locations of the requested coordinate flatfiles (step 3). If the data is stored locally then it is returned otherwise a list of remote data source locations are returned to the Data Retrieval Component (step 6). A data source is then selected from the list and a request is made to the Data Retrieval Component on the remote site for the required data (step 7). As this source is listed as a valid data source it is guaranteed to store the data locally, hence it will not attempt to retrieve the data from another remote site. The data is then passed back to the requesting site (step 10) and the Data Retrieval Component returns the data to the user script (step 11) in the same way as a locally stored data set.

There are two key opportunities to save retrieval times when retrieving large amounts of data. The first is to look at the list of sites that store the trajectory and ask multiple sites to provide different parts of the trajectory. This will reduce the load on

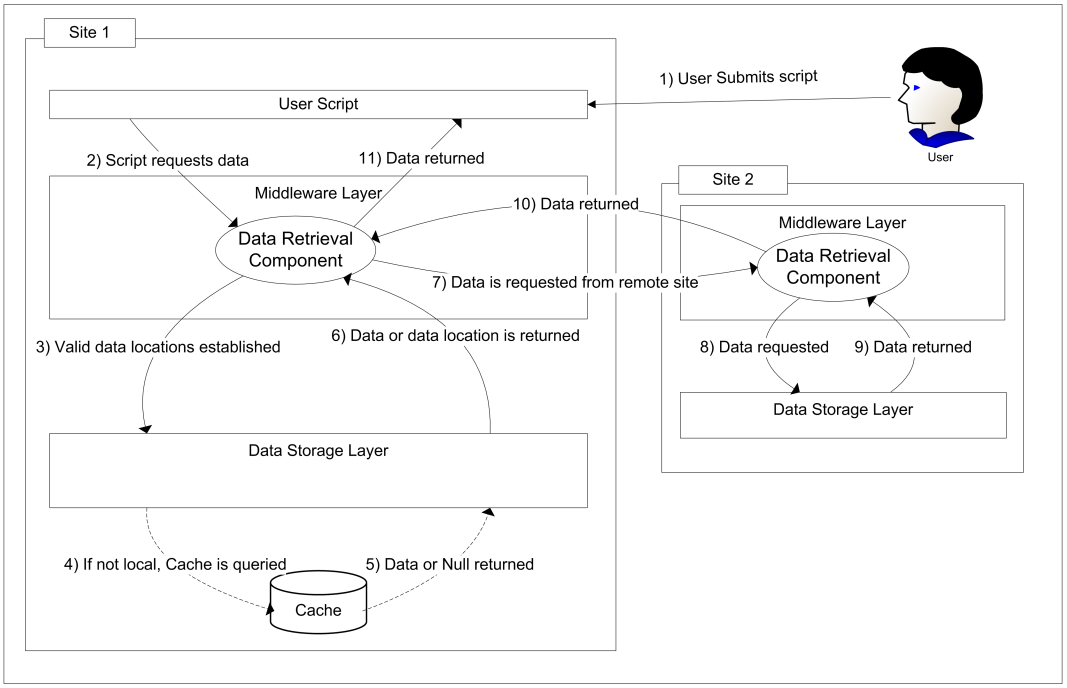

Fig. 5: Schematic showing how a remote data request is returned. Caching mechanism is used to improve the speed of data transfer.

sites by distributing it across multiple sites as well as improving the speed that data is received.

The second is a cache (not implemented in the current prototype). Each frame that is retrieved from an external site will be stored using the same flatfile storage method. If a whole trajectory is then cached it can be moved to the main database and marked as a valid location to retrieve data for that trajectory. So when a data query requires data that is not stored on the database then the cache is consulted first to see if it has been retrieved previously (step 4 and 5 in Figure 5 if not then the hosting site is queried. There is a limit to the number of frames that are held in the cache and this is defined by a site specific limit, which also includes the whole trajectories that are added to the local data store. The aim of storing whole trajectories on additional sites is to attempt to move the data closer to the processing. If a site continually requests a trajectory it makes sense to store the trajectory on that site.

Currently each site has an excess of storage space and we can utilise this space to gain a performance boost however more trajectories can still be added as temporary trajectories can simply be deleted and removed from the metadata database to make more room as required.

The Data Retrieval Component is not only responsible for getting the data from the distributes sources but it is also responsible for making the data transparently available to the users in an environment of their choice, in this case Python. This results in Python numeric arrays being made available to users who have no idea where or how the data is stored. This has currently been implemented and a series of analysis tools for the Post Processing Component have been built on this design. This design also permits extensions for other languages like Perl to assist the users to migrate and utilise the BioSimGrid project.

Post Processing Component

For the post processing component, a set of analysis tools are written for standard and generic analysis on the simulation data, e.g. the calculation of Root Mean Square Derivation (RMSD) and the computation of the average structure and interatomic distances. Each analysis is exposed as a module and the modularity approach enables the tool

```
FC = FrameCollection('2, 100-200')
myRMSD = RMSD(FC)
myRMSD.createPNG()
```

Fig. 6: An example of a user script to run a RMSD analysis using frame 100th to 200th from trajectory 2

set to be extended easily. An example of an analysis script is shown in figure 6 to demonstrate how to use the post processing tools. The first line of the script specifies a frame collection - the part of a protein to be used to perform the analysis, in this case frame 100th to 200th from trajectory 2. The next line requests a RMSD analysis by taking the frame collection as its input parameter. Finally, the third line specifies the format of the result to be generated, which in this case is the output of an image file in PNG format. The ease of selecting different data set and different post processing tools allows biochemists with little computational experience to perform an analysis on the simulation data and obtain meaningful results.

3.3 User Interface Layer

BioSimGrid user application level offers two modes of interaction: via a graphical web based interface or via the Python scripting environment. The graphical interface is just another layer on top of the underlying Python codes. The scripting environment caters for advanced users who would like to connect to BioSimGrid in a scripting environment and utilise its data submission, retrieval and post-processing API in a fully programmable way. In this environment, biochemists can choose to run existing analysis toolkits provided by BioSimGrid. Alternatively for more specific analysis, they can use the available data retrieval packages to write their own script. The graphical interface provides a more user friendly environment to cater for novice users. It allows users to perform standard analysis runs and provides an overview of the available data and processing options. In this mode, a user first selects an analysis from a drop down menu, then proceeds to select a trajectory and the relevant frames to perform the analysis on. All these operations are done by clicks of buttons on a web browser.

4 Current Issues and Next Prototype

BioSimGrid is in its early stage of development. Current prototypes that have been developed are based on architecture where both the application and database server are implemented as client server architecture, running at a single location. We have modularised our components and have developed a basic set of functionalities of BioSimGrid for data deposition, data retrieval and analysis of post simulation data. The modularity approach of the components enables easy plug-in and future extension of various functionalities, such as adding more analysis tools or extending the data deposition tools to cater for new simulation result formats.

The next prototype of BioSimGrid will concentrate on tackling the geographically distributed databases and applications. Establishing secure asynchronous network communication, handling data latency and data recovery is non trivial in this case. We are investigating Python twisted framework [11] and Pyro [12] for programming network services and applications. For a more reliable data transmission, the next prototype will incorporate MD5 [13] hashes to help manage corruptions in file transfer. We also envisage the use of standard protocols such as secure socket layer (openSSL) to provide secure point to point communication.

The issue of security is also a major concern in BioSimGrid. We envisage the use of digital certificate-based authentication to authorise users into the system and provide mechanism to set various permission levels for different user groups to authorise them to different resources and transactions.

In the future work, we plan to implement web service based interfaces in order to provide a platform and language independent way of accessing the existing middleware components.

5 Conclusion

In summary, BioSimGrid provides a trajectory storage system which allows users to submit simulation data from a wide range of simulation packages and to run cross simulation comparisons independent of the source of the data. We have developed the current version of the system together with biochemists who provide constant feedback on the usability of the project, and we are currently expanding the user base and the number of available trajectories in the system.

Acknowledgements

We would like to thank our collaborators D. Moss, C. Laughton, L. Caves, O. Smart and A. Mulholland. This project is funded by BBSRC.

References

1. Bing Wu, Kaihsu Tai, Stuart Murdock, Muan Hong Ng, Steven Johnston, Hans Fangohr, Paul Jeffreys, Simon Cox, Jonathan Essex, and Mark S.P. Sansom. Biosimgrid: a distributed database for biomolecular simulations. In Simon J. Cox, editor, *Proceedings of UK e-Science All Hands Meeting 2003*, pages 412–419, Swindon, 2003. EPSRC.
2. Andrew Dalke William Humphrey and Klaus Schulten. Vmd — visual molecular dynamics. *Journal of Molecular Graphics, 14:33-38*, 1996. http://www.ks.uiuc.edu/Research/vmd/.
3. W.L. DeLano. The PyMOL molecular graphics system. *DeLano Scientific*, 2002. http://www.pymol.org.
4. The GridPP Project. http://www.gridpp.ac.uk.
5. The EU DataGrid Project. http://www.eu-datagrid.org.
6. The OGSA-DAI Project. http://www.ogsa-dai.org.uk.
7. *OGSA-DAI: Two Years On*, GGF10, Data Area Workshop, Humboldt University, Berlin Germany, 2004.
8. Fred L. Drake Jr. Guido van Rossum. Python library reference. *Computer Science Department of Algorithmics and Architecture, CS-R9524*, 1995. http://www.python.org.
9. Unidata - netcdf. http://my.unidata.ucar.edu/content/software/netcdf/index.html.
10. *The Molecular Modeling Toolkit: a case study of a large scientific application in Python*, http://starship.python.net/crew/hinsen/MMTK, 1997.
11. Twisted Matrix Laboratories. http://www.twistedmatrix.com.
12. PYRO - Python Remote Objects. http://pyro.sourceforge.net/index.html.
13. RFC 1321 - The MD5 Message Digest Algorithmn. http://www.faqs.org/rfcs/rfc1321.html.