A Generic Framework and Methodology for Implementing Science Gateways for Analysing Molecular Docking Results

Damjan Temelkovski, Tamas Kiss, Gabor Terstyanszky
University of Westminster, London, UK
damjan.temelkovski@my.westminster.ac.uk, {t.kiss, g.z.terstyanszky}@westminster.ac.uk

***Abstract*—Molecular docking and virtual screening experiments require large computational and data resources and high-level user interfaces in the form of science gateways. While science gateways supporting such experiments are relatively common, there is a clearly identified need to design and implement more complex environments for further analysis of docking results. This paper describes a generic framework and a related methodology that supports the efficient development of such environments. The framework is modular enabling the reuse of already existing components. The methodology is agile and encourages the input and participation of end-users. A prototype implementation, based on the framework and methodology, of a science-gateway-based molecular docking environment for recommending a ligand-protein pair for next docking experiment is also presented and evaluated.**

Keywords—bioinformatics; modelling; molecular docking; science gateway; virtual screening.

I. Introduction

Molecular docking is a computational simulation that models biochemical interactions to predict where and how two molecules would bind. Large-scale molecular docking simulations are used in areas such as drug discovery where they can decrease the amount of wet-lab experiments required. Since molecular docking uses the structure of the receptor, large-scale molecular docking of hundreds of thousands of ligands and one receptor is called structure-based virtual screening (*virtual* as opposed to the robotics-based high throughput screening). Although a single docking simulation is relatively short, a typical virtual screening experiment, that may combine thousands of simulations, is computationally demanding, requiring the use of Distributed Computing Infrastructures (DCIs). Utilising and accessing such computational resources adds an extra level of complexity to the task making it increasingly difficult for biomedical scientists. Science gateways are widely utilised in this area to help bridging this gap.

Although this field has seen great advancements recently, feedback from biomedical scientists shows that there is still a significant gap to bridge. Examples for science gateways supporting molecular docking and virtual screening experiments include several WS-PGRADE/gUSE [1] based gateways, such as the MosGrid Portal [2], the AutoDock Gateway [3], and the AMC Docking Gateway [4]; as well as non-workflow-based pipelines such as the virtual screening environment for Windows Azure [5], the supercomputer-based [6] or the Linux cluster-based [7] virtual screening pipelines. However, there is still a need for more complex environments that enable scientists to access a wide range of computing, data and network resources for the further analysis of docking results. Such environments should support complex scenarios where intelligent support can be provided for the more efficient execution of large-scale molecular docking experiments.

This paper investigates such scenarios and proposes a generic conceptual framework to support the analysis of molecular docking results, and a related methodology that uses regular input from scientists when developing complex science-gateway-based environments for the storage, analysis and reuse of molecular docking results. It has been developed considering biomedical scientists' requirements collected from semi-structured interviews and a literature review of 14 related projects including those mentioned in the paragraph above. From this generic framework, specific architectures can be derived supporting various molecular-docking-related analytical scenarios as shown in Section II. Additionally, a software development methodology that supports creating docking experiments based on this framework is explained in Section III. Finally, a prototype implementation of such system is presented in Section IV.

II. Generic Framework for the Analysis of Molecular Docking Results

The aim of our research was to identify potential similarities in the work of biomedical scientists working with molecular docking experiments, and to investigate whether a generic framework for such application scenarios can be defined. The assumption was that based on this generic framework more specific science gateway based environments can be implemented supporting different application scenarios. As these scenarios have large similarity, deriving and implementing such specific environments can be speeded up significantly. In other words, the aim was to formalise and

This work was partially supported by the COLA Cloud Orchestration at the level of Applications project, Project No. 731574.

speed up the development of specific science gateway environments supporting various molecular docking scenarios.

In order to identify typical user requirements, several interviews with five scientists from different backgrounds and with various degrees of experience with molecular docking simulations were conducted. Since the number of the interviewees was small and the population localised in London, this is a not a representative sample of the world-wide population of scientists that use molecular docking simulations. However, considering its diversity, the sample was useful in producing several conclusions. The interviews aimed at identifying requirements of the scientists when performing molecular docking experiments and specifying scenarios that are not supported by currently available science gateways for molecular docking. These scenarios typically represent software systems that make a decision based on the molecular docking results, mimicking the steps that a scientist needs to take after obtaining the results. Some representative and identified scenarios are listed below:

1. Suggest a ligand-protein pair that should be used in the next molecular docking, based on protein similarity and previous results
2. Filter docking results which are suitable for wet laboratory experiments, based on ligand properties
3. Find off-target drugs, based on deducing if the estimated binding is at an active site
4. Enable verification of the docking methodology and learning from previous docking for novice users
5. Compare results from different molecular docking tools

Based on the conceptual similarities of these scenarios and an extended review of literature, a generic framework has been designed. The design focuses on the similar elements in the scenarios and includes the following components (see Figure 1):

Molecular Docking Environment (MDE): All scenarios include an environment where the molecular docking simulation is executed. It could be as simple as running a single simulation from the command line on a local computer, to more complex such as executing a virtual screening experiment on a DCI. This environment includes the software tool used for the docking itself, and may also include additional elements to connect to a DCI or to provide a high level user interface.

Molecular Docking Results Repository (MDRR): After the execution of the molecular docking, the results need to be stored as previous molecular docking results are needed by various scenarios. The repository should also store information about the final decision made by the whole simulation environment.

Additional Tool (AT): The results which have been stored in the MDRR are then processed by an AT. This is a generic element that describes a tool which takes one or more molecular docking results as input and conducts a calculation. ATs can refer back to other molecular docking results stored in the MDRR, communicate with other ATs, or refer to data stored in an Additional Data Source.

Additional Data Source (ADS): It contains data that is relevant for the final decision and usually is an external database.

Decision Maker (DM): All the information processed from the various ATs is passed to a DM. This element groups and analyses the calculations performed by the ATs in order to make a decision.

The numbers in Figure 1 present the order or flow of events through the different elements:

1. A scientist uses an MDE to conduct the molecular docking and the result is uploaded to the MDRR.
2. The MDRR sends the results to one or more ATs.
3. An AT may communicate with one or more other ATs.
4. An AT may look up data stored in the ADS.
5. An AT may require additional previous molecular docking results as input for its calculation.
6. An AT would provide its calculation results to the DM.
7. The MDRR may use data from the ADS directly.
8. Previous results from the MDRR may be used by the DM
9. The DM may use data from the ADS directly.
10. Once the analysis is complete and the decision is made, it can be passed back to the MDRR.
11. Finally, the decision is passed to the MDE to visualise it.

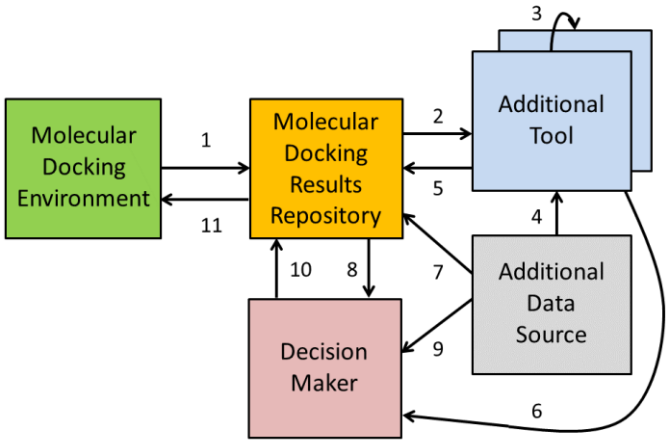

Figure 1 – Basic diagram of the Generic Framework

From this generic framework each specific scenario introduced earlier, and also the ones covered in the literature review can be derived. For illustration, a basic architecture diagram for the first scenario is shown in Figure 2. Similar figures for each scenario have been designed and analysed demonstrating that the framework is generic enough to support at least the five identified scenarios and the 14 related solutions covered in the literature. However, these figures are not presented here due to limitations in length of the paper.

In Scenario 1 (Figure 2) the framework would analyse previous molecular docking results and look for good docking results that have used a receptor similar to the currently used receptor. Based on this, the system suggests a new protein-ligand pair that would be an interesting candidate for docking. Two key issues here are the definitions of *good* docking result or *similar* receptor.

In Figure 2 the building blocks of the Generic Framework have been replaced with concrete elements supporting this particular scenario. One of the advantages of this modular design is that these building blocks can be easily replaced with other elements if necessary. This way multiple existing tools can be integrated into the scenario design and evaluated, requiring only the implementation of components that are not currently available. Mapping of the generic framework for this particular scenario in the presented example is as follows:

The MDE is an extended version of the popular Racoon2 [8] desktop application, a virtual screening environment. The WS-PGRADE/gUSE science gateway framework was integrated with Raccoon2 to support large-scale experiments on heterogeneous cloud computing resources, as it was presented in [9]. The MDRR is a custom-made repository based on a MongoDB database. Three ATs are utilised in this scenario. The structural alignment tool DeepAlign [10] is used to calculate similarities between receptors. A custom-made AT is used to assess whether the structural alignment result means that the two receptors are similar, while another custom-made AT is required to assess a docking result and categorise it as *good*. Finally, a custom-made DM is needed to suggest which protein-ligand pair to dock next.

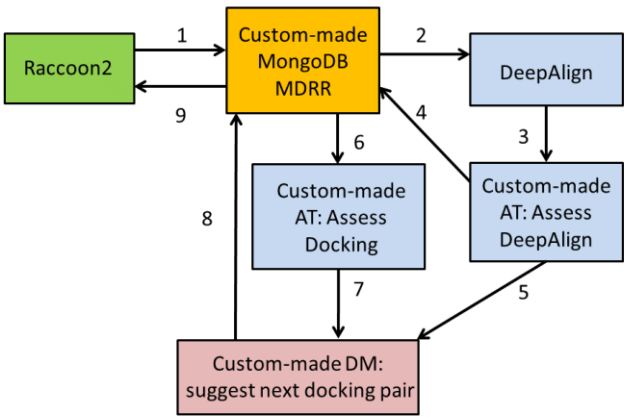

Figure 2 – Basic diagram of scenario to suggest a ligand-protein pair for next docking (Scenario 1)

The flow of events is shown in Figure 2. Raccoon2 executes the molecular docking and the results are uploaded to the MDRR (1). The MDRR sends the receptor pairs to DeepAlign (2). The results of DeepAlign are assessed by the custom-made AT (3) that sends the results to the MDRR (4) and the DM (5). All past docking results of similar receptors are sent to be assessed (6) and the *good* results are sent to the DM. The DM combines the results from the ATs, and suggests which protein-ligand pair to dock as a next step. This suggestion is returned to the MDRR and stored as meta-data (8). Finally, it is presented to the user (9).

Based on the basic generic architecture of Figure 1, a more detailed framework has been developed that consist of a diagram, a textual description of elements and interfaces, and a formal description using Z-notation [11]. The aim of this framework is to describe the generic architecture and the way how the specific scenarios are derived from this in a formalised way. Based on this formalism we aim to support application developers to make specific decisions when evaluating and implementing these scenarios. The designed framework is independent from the actual implementation, or indeed, the programming language of choice.

The diagram representing the framework in Figure 3 is a generic model, showing all generic elements and all possible interfaces between them. It is based on the UML Component Diagram in the sense that the elements are drawn as components and the interfaces between them are the typical *provided* and *required* interface connections. Additionally, it features arrows pointing towards the direction of the flow of data in a particular interface.

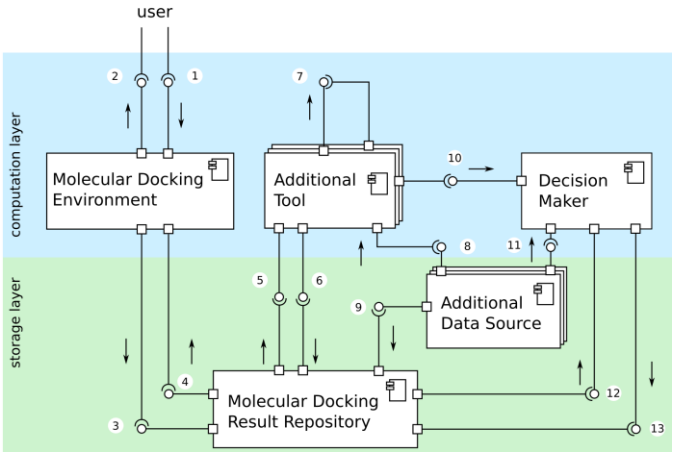

Figure 3 – Generic Framework diagram

The framework features 13 interface types between its elements. As next step, each of these interfaces have been identified and described. For example:

1. User → MDE, provided by the MDE: allows the user to upload the correct input for the molecular docking or additional user input values needed by another element.
2. MDE → user, provided by the MDE: displays the result of the molecular docking to the user, along with other results from the MDRR.

Following this, each element and each interface have been described formally using Z-notation. As the set of descriptions is too extensive for this paper, only a representative example is

presented here, describing the MDE and its interfaces (see Figures 4 and 5).

The docking process expressed by the MDE needs a ligand, receptor, and optionally configuration (config) files as input, and provides a docking result file as output. When there is no config file then the *dockingWithoutConfig()* function will generate the docking result, while when there is a config file then the *dockingWithConfig()* function will do it. Furthermore, the Z-notation for *dockingWithoutConfig()* describes that for every ligand × receptor pair, as long as the ligand and receptor are not empty files, there exists a docking result. Similarly, *dockingWithConfig()* defines that for each ligand and for each receptor there exists a configuration file that can be used to produce a docking result. The corresponding Z-notation descriptions can be seen in Figure 4.

$$dockingWithoutConfig : (LIGAND \times RECEPTOR) \pfun RESULT$$

$$\forall l : LIGAND;\ r : RECEPTOR \mid l \neq \varnothing \land r \neq \varnothing \bullet \exists res : RESULT \bullet dockingWithoutConfig(l, r) = res$$

$$dockingWithConfig : (LIGAND \times RECEPTOR \times CONFIG) \pfun RESULT$$

$$\forall l : LIGAND;\ r : RECEPTOR \mid l \neq \varnothing \land r \neq \varnothing \bullet \exists c : CONFIG;\ res : RESULT \mid c \neq \varnothing \bullet dockingWithConfig(l, r, c) = res$$

Docking

$ligand? : LIGAND$
$receptor? : RECEPTOR$
$config? : CONFIG$
$result! : RESULT$

$$config? = \varnothing \land result! = dockingWithoutConfig(ligand?, receptor?)$$
$$\lor$$
$$config? \neq \varnothing \land result! = dockingWithConfig(ligand?, receptor?, config?)$$

Figure 4 – MDE Described in Z-notation

Figure 5 models the MDE and its interfaces for the three types of input files. This schema explains that the ligand, receptor, and config files are input, while the docking results as well as data about the date are produced as output. The lower part of Figure 5 describes the interface that enables users to view results, as long as they are not non-existent.

MolecularDockingEnvironment

$ligands? : LIGANDS$
$receptors? : RECEPTORS$
$config? : CONFIG$
$results! : RESULTS$
$date! : DATE$

ViewMolecularDockingResults

$\Xi MolecularDockingEnvironment$

$$results! \neq \varnothing$$

Figure 5 – Interfaces of the MDE described with Z-notation

Based on the above detailed description of the generic framework, a detailed architecture diagram of each scenario can now be derived followed by the textual and formal descriptions of these scenarios. Figure 6 shows part of the detailed architecture diagram of Scenario 1, representing the extended Raccoon2 as an MDE, and corresponding to that part of the basic diagram of Figure 2. In Figure 7 the formal description of this module is shown. (Please note that full diagram and description are not provided due to limitation of length, but has been produced.)

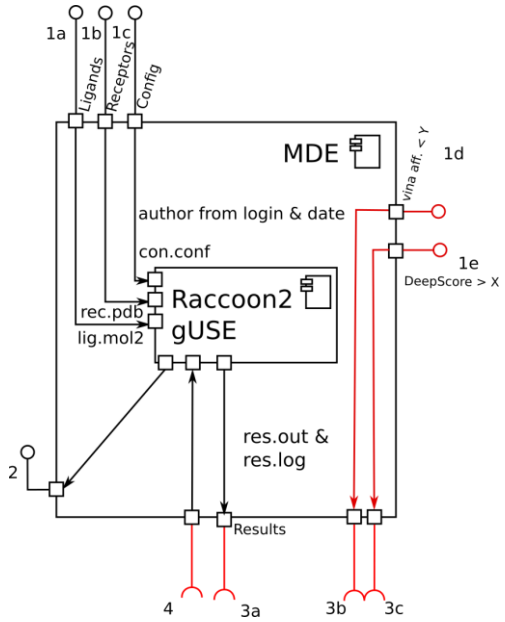

Figure 6 – Extract of the detailed architecture diagram of Scenario 1

$$dockingWithConfig : (LIGAND \times RECEPTOR \times CONFIG) \pfun RESULT$$

$$\forall l : LIGAND;\ r : RECEPTOR \mid l \neq \varnothing \land r \neq \varnothing \bullet \exists c : CONFIG;\ res : RESULT \mid c \neq \varnothing \bullet dockingWithConfig(l, r, c) = res$$

Docking_AutoDockVina

$ligand? : LIGAND$
$receptor? : RECEPTOR$
$config? : CONFIG$
$result! : RESULT$

$$config? \neq \varnothing \land result! = dockingWithConfig(ligand?, receptor?, config?)$$

MolecularDockingEnvironment_Raccoon2

$ligands? : LIGANDS$
$receptors? : RECEPTORS$
$config? : CONFIG$
$results! : RESULTS$
$date! : DATE$

$$\exists ligand? : ligands?;\ receptor? : receptors?;\ result! : results! \bullet Docking\_AutoDockVina$$

Figure 7 – Extract of the formal description of Scenario 1

III. Methodology for Developing Environments for the Analysis of Molecular Docking Results

This section describes the methodology for developing complex environments that reuse and analyse molecular docking results. This methodology complements the framework described in the previous section by explaining how this framework can be used during development. It clearly states the roles that are required and the specific sub-

projects for which they need to collaborate. The methodology is based on the seven principles identified by Cockburn [12].

Based on Cockburn's general recommendations, a *role-deliverable-milestone* diagram has been created to represent the methodology (Figure 8). This diagram illustrates that the modeller, biomedical scientist and bioinformatician should collaborate when creating the diagram and textual description of the scenario. Furthermore, the modeller should collaborate with the bioinformatician and the software developer when creating the formal description. Key components of this diagram, extensions to Cockburn's original model, are the dotted lines which show that the process is agile. For instance, in the top section where the life scientist works on the textual description and go from milestone M4 to M5, there is a dotted line showing that (s)he could revisit and alter the diagram if necessary. The same logic is used for the agile development of the final system code. Figure 8 presents a high level role-deliverable-milestone diagram where the coding section has an asterisk (*) indicating that a similar but more detailed description of this section (not presented in this paper) has also been developed in the form of a lower-level diagram.

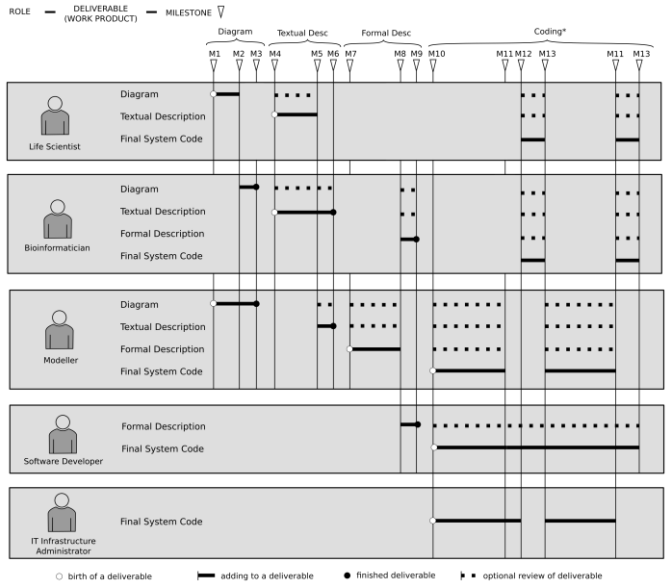

Figure 8 – Role-deliverable-milestone diagram of the developed methodology

IV. Implementation of the Selected Scenario

In order to demonstrate how the developed framework and methodology support implementing molecular docking science gateways, an implementation of Scenario 1 (https://github.com/damjanmk/mdrr-scenarios) is presented here. All components in the implementation are accessible via a basic RESTful API. We used *Bottle* [13], a minimalist web-framework which enables easy server setup. The MDRR and the DM have been deployed on Server 1, the DeepAlign AT and the AT to assess the DeepAlign results on Server 2, while the docking assessment AT on Server 3 (Figure 9). In order to insert results from Raccoon2, the MDRR on Server 1 expects zip files as POST parameters. It parses them and inserts information into MongoDB, which includes the collections *receptors*, *ligands*, *results*, and *analysis*. Another request is sent to continue with Scenario 1 where the MDRR selects *all receptors* from the database, parses and compresses them. Next, these are sent to Server 2 along with the *target receptor* (the receptor used in the original simulation), and a *threshold* value (input by the user in Raccoon2). The first AT on Server 2 executes DeepAlign to find similarities between the target receptor, and each different receptor it received. It then calls the AT: AssessDeepAlign, located on the same server, in order to select the similar receptors. In the simplest form of this AT, it assesses the DeepAlign results by comparing the value of *DeepScore* to a user input threshold. A list of these similar receptors is returned to Server 1 where the *analysis* collection is updated to keep track of the events so far. Then, the MDRR selects past docking results which have used one of the similar receptors, and compresses them. It sends a request to Server 3, including a threshold value of the AutoDock Vina *affinity*, entered by the user within Raccoon2.

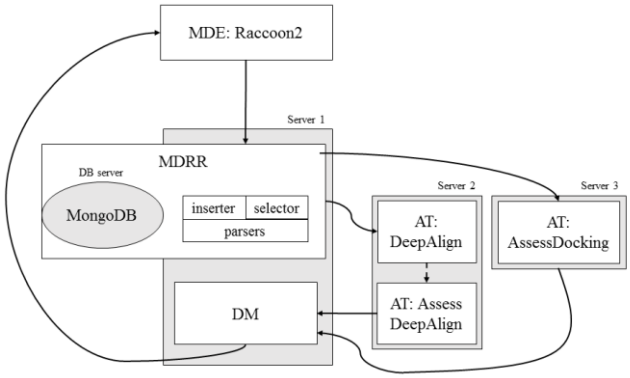

Figure 9 – Architecture of implementation of Scenario 1

The AT on Server 3 searches through the Vina results for a result that has at least one model where the Vina affinity is less than the threshold, and calls this a good docking result (a Vina docking result can contain for example 10 models). It returns a list of good docking results to Server 1.

Upon receiving this, Server 1 inserts a document in the analysis collection before initialising the DM and sending it the *similar receptors* and the *good results*. The DM combines these two lists into one and sorts it based firstly on the DeepScore value, then on the affinity. This enables users to view an ordered list of results that contain ligands which are suggested for a subsequent docking.

A. Designing the MongoDB database

At the core of this custom-made MDRR is a MongoDB database. There were several reasons why we chose this type of non-relational database:

1. MongoDB's schema-less design is ideal because a single collection can be used for: input files in different formats, output files of any of the over 50 docking tools [14], or meta-data about different ATs.

2. MongoDB scales very well for large amounts of data, provided it is well designed and features such as sharding and indexing are utilised.
3. MongoDB is well-suited for prototyping because it is easier to change what is stored during development.

In this prototype implementation we have considered *.pdbqt* molecules and AutoDock Vina results (as used by Raccoon2).

The ligands collection contains molecular properties calculated using the OpenBabel and PyBel [15] Python modules such as *canonical_SMILES*, *logP*, *mol_weight*, etc. Biomedical scientists at the University of Westminster were consulted when deciding which properties to store. Both the ligands and receptors collections include the full parsed 3D structure from the .pdbqt files. Each line of the .pdbqt file is stored as an element of an array. The structure of each molecule should be unique. However, the structure itself cannot be uniquely indexed due to size limitations, so we have introduced *structure_id* - an MD5 hash of the structure. This uniquely describes the structure and allows for a MongoDB index to be created.

The results collection contains references to the ligand and receptor used, specific properties extracted from the result files (e.g. *CPUs*, *random_seed*), a list of the result *models*, each model containing *affinity*, *rmsd_from_best*, and the parsed model segment of the Vina result. The parsing process is simple – it stores all lines between *MODEL* and *ENDMDL* as elements of an array.

B. Use of the framework and methodology

The framework was followed as described in Section II. A list of documented meetings and events is not presented with this paper, but serves as supporting evidence of following the methodology. The required roles were taken up by different researchers at the University of Westminster (with some doubling as multiple roles). The presented implementation proves that following the methodology such molecular docking framework can be implemented. Work is currently ongoing to quantify advantages when compared to more ad-hoc implementation.

C. Limitations of the prototype implementation

Due to the Global Interpreter Lock (GIL), Python is not the optimal language for multi-threading without additional optimisations. Furthermore, Bottle uses a non-threading type of servers by default, so using a different specialised server would improve performance for simultaneous users. The number of items in the collections may become too big to be included in one zip file which is used to transfer data from servers and sending large files through the network could be a bottleneck. Finally, the current DM joins and sorts two lists without specific performance optimisations.

V. Conclusion and Future Work

This paper presented a generic framework and a corresponding methodology to implement complex science-gateway-based environments for the execution of molecular docking experiments extended with the intelligent analysis and utilisation of docking results. The framework incorporates a diagram, and textual and formal description enabling a modular design and the replacement and reuse of components. The methodology involves multiple stakeholders and requires their collaboration in an agile manner. In order to demonstrate the usability of the above, a scenario for suggesting a ligand-protein pair for next docking was also presented.

Future work includes the implementation and detailed evaluation of multiple scenarios to identify, and where possible quantify, the advantages provided by the framework and methodology. In order to achieve this, the implemented solutions are compared to state-of-the-art methods and environments to demonstrate the added value of our research.

References

[1] P. Kacsuk et al., "WS-PGRADE/gUSE generic DCI gateway framework for a large variety of user communities" *J. Grid Comput.*, vol. 10, no. 4, pp. 601-630, Dec, 2012.

[2] J. Krüger et al., "The MoSGrid Science Gateway – A Complete Solution for Molecular Simulations", *J. Chem. Theory Comput.*, vol. 10, no 6, pp. 2232–2245, May, 2014.

[3] Z. Farkas et al., "AutoDock gateway for user friendly execution of molecular docking simulations in cloud systems", in *Cloud Computing with E-science Applications*, Olivier Terzo, Lorenzo Mossucca, Eds. Boca Raton, FL: CRC Press/Taylor & Francis, 2015, pp 217-236.

[4] M. Jaghoori et al., "A multi-infrastructure gateway for virtual drug screening", *Concurr. Comp. Pract. E.*, vol. 27, no. 16, pp. 4478–4490, Nov, 2015.

[5] T. Kiss et al., "Large-scale virtual screening experiments on Windows Azure-based cloud resources", *Concurr. Comp. Pract. E.*, vol. 26, no 10, pp. 1760-1770, Jul, 2014.

[6] X. Zhang, S. E. Wong, and F. C. Lightstone, "Toward fully automated high performance computing drug discovery: a massively parallel virtual screening pipeline for docking and molecular mechanics/generalized born surface area rescoring to improve enrichment", *J. Chem. Inf. Model.*, vol. 54, no 1, pp. 324-337, 2014.

[7] P. D'Ursi et al., "Virtual screening pipeline and ligand modelling for H5N1 neuraminidase", *Biochem. Biophys. Res. Commun.*, vol. 383, no. 4, pp. 445-449, Jun, 2009.

[8] S. Forli et al., "Computational protein-ligand docking and virtual drug screening with the AutoDock suite". *Nat. Protoc.*, vol. 11, no. 5, pp. 905-919, Apr. 2016.

[9] D. Temelkovski, T. Kiss, and G. Terstyanszky, "Molecular docking with Raccoon2 on clouds: extending desktop applications with cloud computing", in *9th International Workshop on Science Gateways*, Poznań, Poland, 2017.

[10] S. Wang, J. Ma, J. Peng, and J. Xu, "Protein structure alignment beyond spatial proximity", *Sci Rep* vol. 3, no. 1448, Mar, 2013.

[11] J. M. Spivey, *The Z Notation - A Reference Manual*, 2nd ed. Oxford, UK: Oriel College, 1998.

[12] A. Cockburn, *Agile Software Development: The Cooperative Game*, 2nd ed. Boston, MA: Addison Wesley, 2006.

[13] M. Hellkamp. *Bottle: Python Web Framework* [Online]. Available: https://bottlepy.org/docs/dev/. [Accessed: 6 Mar 2018]

[14] Swiss Institute of Bioinformatics. *Directory of in-silico Drug Design tools - Docking* [Online]. Available: https://www.click2drug.org /directory_Docking.html. [Accessed: 6 Mar 2018]

[15] N. M. O'Boyle, C. Morley, and G. R. Hutchison, "Pybel: a Python wrapper for the OpenBabel cheminformatics toolkit", *Chem. Cent. J.*, vol. 2, no. 5, Mar 2008.