Extending Molecular Docking Desktop Applications with Cloud Computing Support and Analysis of Results

Temelkovski, D., Kiss, T., Terstyanszky, G. and Greenwell, P.

Extending Molecular Docking Desktop Applications with Cloud Computing Support and Analysis of Results

Damjan Temelkovski[*], Tamas Kiss, Gabor Terstyanszky, Pamela Greenwell

University of Westminster, Faculty of Science and Technology, 115 New Cavendish Street, London, W1W 6UW

Abstract

Structure-based virtual screening simulations, which are often used in drug discovery, can be very computationally demanding. This is why user-friendly domain-specific web or desktop applications that enable running simulations on powerful computing infrastructures have been created. This article investigates how domain-specific desktop applications can be extended to use cloud computing and how they can be part of scenarios that require sharing and analysing previous molecular docking results. A generic approach based on interviews with scientists and analysis of existing systems is proposed. A proof of concept is implemented using the Raccoon2 desktop application for virtual screening, WS-PGRADE workflows, gUSE services with the CloudBroker Platform, the structural alignment tool DeepAlign, and the ligand similarity tool LIGSIFT. The presented analysis illustrates that this approach of extending a domain-specific desktop application can use different types of clouds, thus facilitating the execution of virtual screening simulations by life scientists without requiring them to abandon their favourite desktop environment and providing them resources without major capital investment. It also shows that storing and sharing molecular docking results can produce additional conclusions such as viewing similar docking input files for verification or learning.

Keywords: bioinformatics, cloud computing, molecular docking, Raccoon2, virtual screening, WS-PGRADE/gUSE

1. Introduction

Biochemical interactions between two molecules can be estimated using a software simulation technique known as molecular docking. Particularly important in drug discovery, this technique can predict the conformation, pose, and

[*]Corresponding author

Email addresses: `damjan.temelkovski@my.westminster.ac.uk` (Damjan Temelkovski), `t.kiss@westminster.ac.uk` (Tamas Kiss), `g.terstyanszky@westminster.ac.uk` (Gabor Terstyanszky), `p.greenwell@westminster.ac.uk` (Pamela Greenwell) binding affinity of a ligand and a receptor. In order to achieve this, the 3D structure of both molecules must be known. This structure can be experimentally determined (using X-ray crystallography or NMR spectroscopy), or estimated (using homology modelling) [1]. Molecular docking consists of an algorithm to handle the atomic spacial arrangement possibilities, and a scoring function to estimate the energy between a particular arrangement of the ligand and the receptor. Since molecular docking uses the structure of the receptor, large-scale molecular docking between hundreds of thousands of ligands and one receptor is called structure-based virtual screening (virtual, as opposed to the robotics-based high throughput screening). In the remainder of this article, these terms will be abbreviated to *docking* and *Virtual Screening (VS)*. Although a single docking simulation is relatively short, a VS experiment is computationally demanding, requiring the use of Distributed Computing Infrastructures (DCIs).

A DCI can consist of cloud computing resources. Cloud computing is a paradigm based on virtualisation which "enables ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction" [2]. The scalability and elasticity makes it useful for various VS simulations. Clouds are available on-demand and users are typically charged on a pay-per-use basis. This can make scientific applications, such as VS, more accessible for scientists around the world, lowering the cost of using complex computing infrastructure. Scientists and students without access to expensive DCIs, and without expertise in DCI configuration, will be able to run VS easily.

Scientific workflow systems such as Taverna [3], Kepler [4] or WS-PGRADE [5], provide a convenient way to represent and develop complex applications composed of multiple steps and executables. A user-friendly interface usually provides convenient workflow management facilities. In some cases, science gateways are developed, providing a user-friendly way to run workflows. There are several examples of science gateways that use workflows to run VS simulations [6–8]. However, all of these solutions require life scientists to become familiar with new, typically web-based user interfaces, and significantly restrict the use of the docking software for the sake of simplicity and ease of use. On the other hand, there are popular desktop applications which offer greater flexibility, such as Raccoon2 [9]. Unfortunately, these desktop applications are either restricted to local resources, or require expensive compute clusters and significant IT support to run them on DCIs. Such tools typically cannot utilise cloud computing resources.

A generic approach to extend domain-specific desktop applications to execute workflows on clouds, while retaining the same familiar Graphical User Interface (GUI) presented to end-users, is described in this article. The utilisation of distributed heterogeneous clouds to run VS is demonstrated, noting that various other DCIs can also be used with few or no changes to this approach. Additionally, the way this extended desktop tool can be utilised when supporting more complex VS scenarios, is shown. Based on semi-structured interviews with life scientists, five representative scenarios were identified. These scenarios require sharing and storing docking results, and include additional calculation or analysis of the results. While some existing solutions enable additional analysis of results [7, 8, 10], their implementations are specific for the tool-chain of choice and scientists can only access their own stored molecular docking results. In comparison, the presented solution is generic and enables data sharing between scientists. We demonstrate how our extended Raccoon2 environment can form the basis of such complex environments by presenting and evaluating the implementation of a sample scenario.

The rest of this article is structured as follows. Section 2 outlines related work. The generic concepts for enabling a desktop application to use cloud computing, and creating a complex scenario which uses previous docking results are shown in Section 3. A prototype implementation is provided in Section 4, while the results and evaluation are presented in Section 5. Section 6 adds conclusions and future work. This article is a significant extension of a conference paper on this topic [11].

2. Related Work

Related work is provided from two different angles. First, we outline how cloud computing has been utilised by other authors to run VS experiments. Second, we describe work that aims to provide further analysis of these docking results utilising various DCIs.

Applying cloud computing for VS experiments is still relatively new with a limited number of examples. One such example is wFReDoW [12], a web-based environment for docking with AutoDock4 [13]. The user uploads the input files to a server which prepares them for docking, while a custom-made middleware called FReMI is used to run the docking on the Amazon EC2 cloud [14]. The ligand *triclosan* with Protein Data Bank (PDB [15]) ID: 1P45A, and 3,100 snapshots of a receptor of the bacteria *Mycobacterium tuberculosis* generated by Molecular Dynamics (MD) simulations have been run on 5 *c1.xlarge* (8-cores, 7GB RAM, 1.65TB storage) instances.

Another example is AutoDockCloud [16], which uses Hadoop to run AutoDock4 on a private cloud which can conduct 570 docking simulations simultaneously on 57 16-core nodes. Their custom-made scripts prepare the docking, and submits the needed files to the private cloud. To test their solution, the authors used the human *oestrogen receptor alpha* (PDB ID: 1L2I) and 2,637 ligands from the DUD [17] database.

In a third example [18], a custom-made .NET desktop application was developed to submit AutoDock and AutoDock Vina [19] jobs to the Microsoft-Azure-based VENUS-C cloud computing service. The user can upload prepared input files, set the cloud configuration, submit and visualise the results from within this application. A receptor and 10,000 ligands have been docked using 20 *extra small* Azure instances (1-core, 768MB RAM, 20GB storage), for a total of 110,000 CPU hours and more than 40,000 docking runs.

A fourth example uses WS-PGRADE/gUSE technology and the CloudBroker Platform to enable docking with AutoDock4 or AutoDock Vina on various clouds [20]. The user can upload input files, run pre-defined workflows and download the results. This approach uses the web-based WS-PGRADE portal as a user interface which, according to feedback we obtained from scientists, is more restrictive than the richer user interface of an existing desktop application called Raccoon2.

All these attempts provided their own restricted GUI for running the simulations, and required a separate environment for pre-processing or post-processing. They were, with the exception of [20], focusing on one specific cloud computing infrastructure. In comparison, the approach suggested in this article, enables scientists to use the GUI of a popular domain-specific desktop application they are familiar with (Raccoon2). Pre-docking and post-docking facilities to prepare input files and analyse results are provided within the same environment. Finally, our approach utilises a set of services that support a wide range of different clouds.

Most VS experiments on DCIs that provided further analysis of docking results have used CPU and GPU clusters [10, 21–24], grid computing resources [7, 8], or both [6]. They can be divided into VS pipelines, and workflow-based docking systems.

VS pipelines contain a set of scripts or tools, and are usually set up in order to explore a particular molecular interaction. For instance, after preparing the input files and conducting the docking using AutoDock, D'Ursi *et al.* (2009) [10] store the results in a MySQL database, and provide methods for target-specific filtering, energy-threshold-based filtering, and creating schematic diagrams of the ligand-receptor complex. Jiang *et al.* (2008) [23] store the result files on the file system, and provide a method for 3^{rd} party re-scoring and hierarchical clustering. Other examples include [18, 24–26].

Workflow-based docking systems use workflow management systems to define workflows which prepare the docking input files, utilise a docking algorithm, and process the docking results. For instance, Kiss *et al.* (2010) [8] provide workflows to prepare input files, conduct docking simulations, and additionally analyse the docking results by running MD simulations and visualising them. Krüger *et al.* (2014) [7] provide workflows for docking, they store the docking results using a custom-made implementation on a distributed file system, and further analyse the docking results by re-scoring or visualising them. Other examples include [20] and [6].

Extensive research in docking has produced many excellent systems. Some of them merely provide an environment to conduct the docking simulations [6, 18, 20, 26], while others provide additional methods for the user to analyse their results [7, 8, 10, 23–25]. However, in all cases it is only the user's own results that are available for analysis. These systems lack the ability to use previous docking results created by other users, in order to produce new conclusions.

3. Generic Concepts for Extending Docking Applications

This section presents two generic concepts. The first concept describes extending desktop applications with cloud computing support. The second concept describes extending a docking environment to allow further analysis of docking results. Section 4 shows how an implementation of the first concept can be an element of an implementation of the second concept.

3.1. Extension of Desktop Environments

Accessing heterogeneous cloud computing resources from existing desktop applications should be achieved without major re-engineering of the desktop application, or further burdening the end-user. End-users should be able to design and execute the experiments in the same way they have done earlier, but with the possibility to send the computations to cloud computing resources. In order to achieve this, a set of services (we call them Cloud Access Services - CAS) can be invoked from the desktop application. CAS should be available from an API in order to facilitate its integration to the GUI of the desktop application. Additionally, CAS should provide access to a wide range of cloud computing resources. The integration requires two major steps from the developers, as illustrated in Figure 1. During the first step, the CAS are configured to run the application in the cloud. This step typically requires preparing workflow applications describing the experiment, and configuring the CAS to interface with the desired cloud resources. In the second step, minor modification of the desktop application is required in order to facilitate the execution of the workflow and retrieval of the results. Instead of implementing the core component of this concept - the CAS, from scratch, existing tools to support workflow creation and interfacing with cloud computing resources can be applied. This approach speeds up the development and has the potential to result in a mature and highly reliable solution.

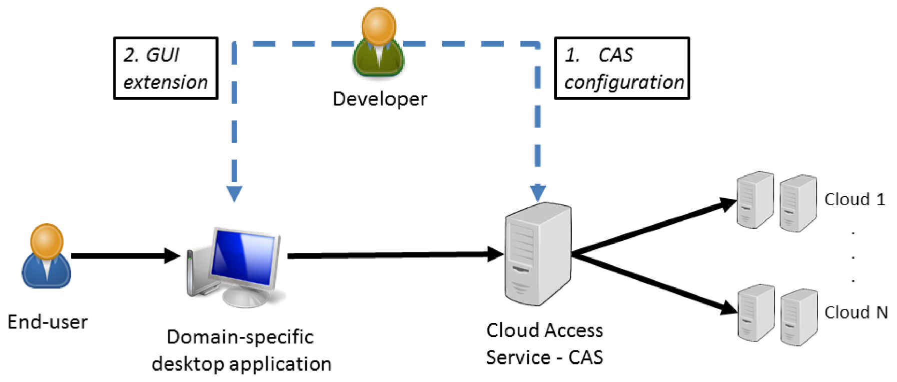


Figure 1: Generic concept for extending desktop applications to run on clouds.

3.2. Further Analysis of Molecular Docking Results

The aim of our work was to identify scenarios where life scientists reuse and further utilise molecular docking results and to investigate how these scenarios can be generalised. We have conducted interviews with London-based scientists and produced a set of scenarios that would require a shared repository of molecular docking results. The following five representative scenarios were identified.

1. *Recommend a ligand-receptor pair that should be used in the next molecular docking, based on protein similarity and previous results.* After a docking simulation has been completed, a new ligand-receptor pair may be needed. The scientist may not have the necessary information about the receptor, so they would search for a similar receptor that has been successfully docked before. The ligands that have been successfully docked to a similar receptor are good candidates for docking with the original receptor.

2. *Filter results based on ligand properties.* When scientists conduct large-scale molecular docking simulations, the results need to be filtered. A scientist wants to identify ligand-receptor pairs to further examine in wet-lab experiments. This may include a ligand which is a good candidate but cannot be examined in the laboratory because of its properties. Therefore, an external database with ligand properties should be consulted.

3. *Find off-target drugs by deducing if the binding is on an active site.* Off-target drugs are drugs that were designed to bind to a receptor, but additionally bind to another receptor of the same or a different organism. In order to find off-target drugs one needs to: find out if the drug binds to the active site of one or potentially a large range of proteins that are not the primary target, look for drugs that may have the same effect, and conduct wet-lab experiments. The first step requires a method to estimate if a docking between a ligand and protein is on an active site. Analysing previous docking results can be used to provide this. If a receptor has been docked multiple times in a certain area, it may be deduced that the area is an active site.

4. *Enable verification of the docking methodology and learning from previous docking.* Storing molecular docking results and their provenance, even if they are "negative" (a.k.a. null) results is important because another scientist may run the same molecular docking simulation and expect to get useful results. They may suspect that they are conducting the docking wrongly. Comparing their input and output files with ones from another scientist will enable them to verify their docking methodology. Furthermore, scientists with little or no experience in running molecular docking simulations can learn more rapidly if they view previous docking input and output files.

5. *Compare results from different docking tools.* Comparing the docking results of the same input files, but using a different molecular docking tool is significant. This will enable scientists to determine if there is a significant difference in the results when using different tools, which may be relevant for further wet-lab experiments.

Based on these scenarios and analysis of existing environments for VS, the elements of a system that uses a repository of shared previous docking results can be generalised. A key component of this concept is the Molecular Docking Environment (MDE) that supports preparation and conducting of docking simulations. An extended desktop environment, as described in Section 3.1, can be an MDE. The docking results storage system is the Molecular Docking Results Repository (MDRR). The tools used to further analyse the results are Additional Tools (ATs), an external database is called Additional Data Source (ADS), and a tool that groups and analyses calculations done by the ATs in order to make a decision is called a Decision Maker (DM). Figure 2 illustrates the flow of events through these different generic elements. This flow is generic for all scenarios:

1. A scientist uses an MDE to conduct the 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 other ATs.
4. An AT may look up data stored in an ADS.
5. An AT may require additional previous molecular docking results as input for its calculation.
6. An AT sends its calculation results to the DM.
7. The MDRR may use data from an ADS directly.
8. The DM may use previous results from the MDRR.
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. The MDE receives the decision and presents it to the user.

In Section 4 we will illustrate how this generic concept can form the basis of an implementation of a specific scenario, and how this modular approach enables reusing existing components or integrating custom developed tools.

4. Implementation of Proposed Concepts

Based on the generic concepts described in Section 3, a reference implementation has been developed. The tools used in this implementation are outlined in Section 4.1. When implementing the generic concept of Figure 1, the domain-specific desktop application is Raccoon2; the CAS is composed of a gUSE server connected to the CloudBroker Platform, a WS-PGRADE portal for workflow development, and the CloudBroker web interface for deployment; while the cloud infrastructure is the University of Westminster (UoW) OpenStack cloud or the CloudSigma cloud [27] (Figure 3). Section 4.2 provides details about the implementation of the Raccoon2 extension. This cloud-enabled extension of Raccoon2 is an MDE in a scenario that retrieves previous docking results with similar input files in order to help the user verify their docking methodology or learn from previous experiments (Scenario 4 from Section 3.2). Please note that

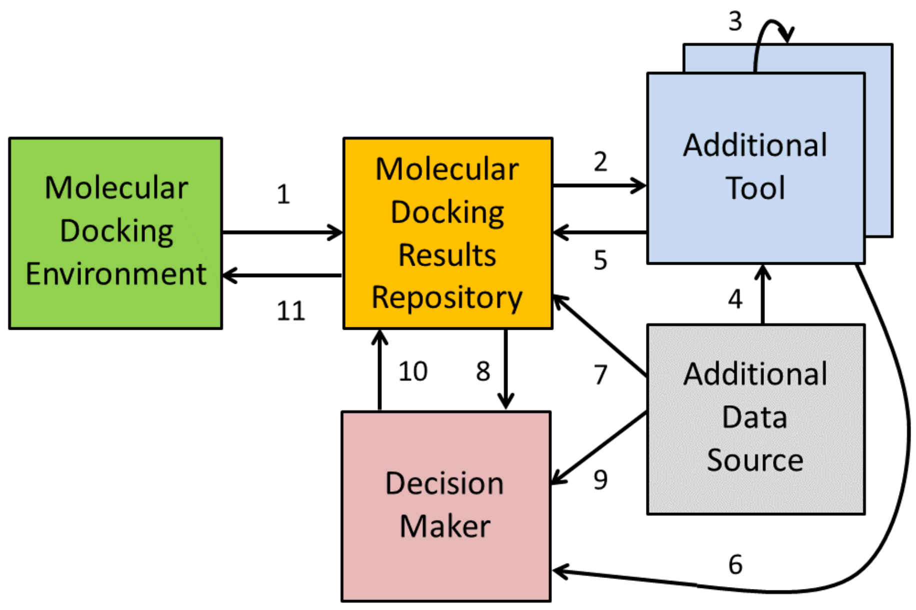

Molecular Docking Environment
Molecular Docking Results Repository
Additional Tool
Additional Data Source
Decision Maker
1
2
3
4
5
6
7
8
9
10
11

Figure 2: Generic concept for further analysis of previous docking results.

although we only present the implementation of this specific scenario in this article, all other identified (and also other similar) scenarios can be (and in some cases have been) implemented using the generic concept in a modular way where components can be easily reused. In the implementation of Scenario 4, which is based on Figure 2, structurally similar receptors and ligands are identified using the ATs DeepAlign and LIGSIFT. Furthermore, three custom-made ATs have been developed: the first assesses the results of DeepAlign, the second assesses the results of LIGSIFT, and the third compares AutoDock Vina configuration files. The implementation includes a custom-made DM, and a MongoDB-based MDRR. Section 4.3 provides details about the implementation of this scenario. The source code of the extended version of Raccoon2 and the implementation of Scenario 4 is available [1,2].

4.1. Background

4.1.1. DeepAlign and LIGSIFT

DeepAlign [28] is a structural alignment tool which compares two receptors based on their structure. It also uses evolutionary information to identify a set of related fragment pairs and produce a score known as *DeepScore*. DeepAlign is the algorithm behind the RaptorX structural alignment web server[3].

[1]`https://github.com/damjanmk/Raccoon2`
[2]`https://github.com/damjanmk/mdrr-scenarios`
[3]`http://raptorx.uchicago.edu/DeepAlign/submit/`

LIGSIFT [29] calculates shape-based, size-independent alignment of small molecules using the scaled Tanimoto Coefficient (sTC). The value for the shape-based sTC is called *ShapeSim* in the LIGSIFT result file.

4.1.2. Raccoon2 and AutoDock Vina

Raccoon2 is a desktop application for preparing and analysing VS simulations. It supports executing docking experiments on a Linux cluster with the PBS or SGE schedulers, and it incorporates analysis features such as filtering, visualising, and exporting results. Users can configure docking options, visualise a binding site, and connect to a cluster to submit jobs, directly from the Raccoon2 GUI. Before the jobs can be submitted, Raccoon2 guides users to deploy the docking tool AutoDock Vina on a computing cluster.

AutoDock Vina is a docking tool that uses a hybrid global-local conformation search with a gradient-based local optimisation. Its scoring function is based on empirically weighted parameters such as: hydrophobic (van der Waals) interactions, hydrogen bonding, and torsional penalties, similarly to its sister-tool AutoDock. AutoDock Vina has built-in support for multi-threading.

Although the proposed concepts of this paper are independent of the docking software used, their evaluation via a prototype implementation requires a particular docking tool. There are more than 50 different docking solutions that may be used for this purpose [30]. Besides AutoDock Vina these include, for example, DOCK [31], GOLD [32], and FlexX (now part of LeadIT) [33]. Sousa et al., [34] have analysed the number of citations of 22 different molecular docking tools and concluded that AutoDock is the most cited docking tool. AutoDock Vina is the latest version of this popular software family. Additionally, AutoDock Vina is embedded into the powerful desktop environment of Raccoon2 making it a suitable candidate to prove our concepts when extending molecular docking desktop applications with cloud computing support. It is still worth emphasising that the concepts are tool independent and can also be applied with different docking software.

4.1.3. WS-PGRADE/gUSE

WS-PGRADE/gUSE [35] is a workflow-based science gateway framework. WS-PGRADE workflows are dataflow directed acyclic graphs where nodes represent execution blocks which can be executed in parallel, and edges correspond to data flow between input and output ports. WS-PGRADE workflows support parameter sweep applications, where a workflow node can be executed many times for multiple input data sets.

A WS-PGRADE portal is an e-science web portal for development of parallel applications using WS-PGRADE workflows and executed on various DCIs. It allows creating, configuring and executing workflows using gUSE (grid and cloud User Support Environment) services. The gUSE is a service stack that can form the back-end of science gateways. It provides well-defined services for workflow management and supports parallel execution on service grids, desktop grids, clusters, and clouds. A gUSE internal component called DCI Bridge [36], provides a well-defined communication interface enabling access to many different DCIs, including clouds [37].

The gUSE RemoteAPI is an API that allows remote submission and management of WS-PGRADE workflows. With it, existing applications can use gUSE services over HTTP(S) without requiring a WS-PGRADE portal. The RemoteAPI requires a valid and well-parameterised WS-PGRADE workflow. It submits this workflow as a temporary user and once downloaded, the workflow output files and all information about this user are deleted from the gUSE server.

4.1.4. CloudBroker Platform

The CloudBroker Platform [38] is a cloud computing middleware and an application store developed by CloudBroker GmbH. It provides a web interface which can be used to deploy and execute an application in a cloud, and monitor its behaviour. The CloudBroker Platform is connected to various types of clouds, including public (e.g. CloudSigma, Amazon Web Services) and private (e.g. based on OpenNebula or OpenStack). The CloudBroker Platform has been integrated with gUSE's DCI Bridge, providing access to various cloud computing resources.

4.2. Extending Raccoon2 with Cloud Support

As described in Section 3.1, the extension of a desktop application has two major steps: configuration of the CAS (1) and modification of the desktop GUI (2). First, the CAS is prepared to execute the VS experiment which includes creating the required WS-PGRADE workflow and configuring the CloudBroker Platform. When accessing gUSE through the RemoteAPI, a valid and well-configured WS-PGRADE workflow needs to be attached. This can be simplified by creating the workflow using a WS-PGRADE portal, testing and then exporting it. The exported workflow files can be manipulated programmatically and attached to a RemoteAPI call. To conclude step 1, the executable files that are needed to run the workflow should be deployed to the cloud, using the CloudBroker Platform. In step 2, the source code of Raccoon2 is extended, in order to configure the workflow and make the appropriate RemoteAPI calls.

4.2.1. Creating the WS-PGRADE Workflow

The execution steps of Raccoon2 are recreated using a WS-PGRADE workflow. In this particular case a simple one-node workflow with four input (ligand files, receptor file, docking configuration file, and an additional file to overcome an output names issue) and one output (the compressed results from the multiple docking runs) ports were created.

Once submitted, the workflow invokes CloudBroker's execution script which runs AutoDock Vina for each ligand it receives as input. In order to run this workflow on many cloud instances, the code of the extended Raccoon2 splits the set of ligands into as many cloud instances as needed, and submits a separate workflow to each instance.

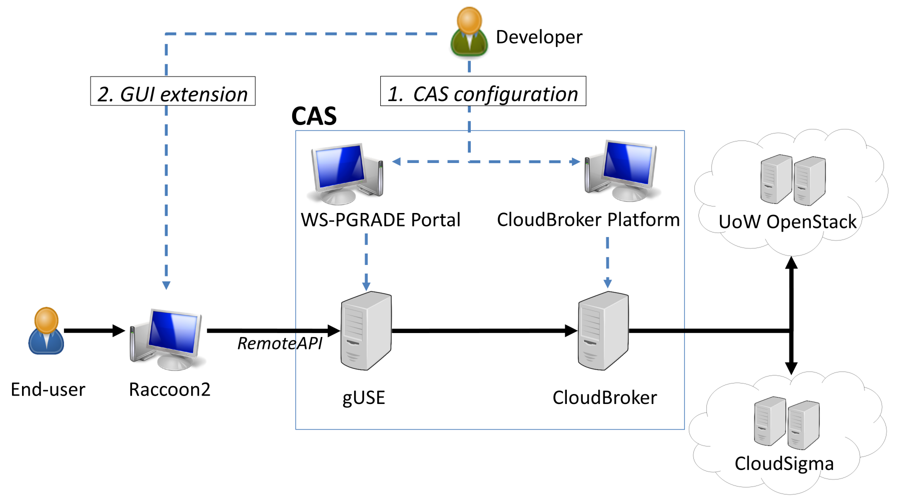


Figure 3: Cloud-enabled Raccoon2 using WS-PGRADE/gUSE, CloudBroker, and two clouds.

4.2.2. Deployment on the CloudBroker Platform

The CloudBroker deployment process requires a deployment script and an execution script. They need to be uploaded and executed on the CloudBroker Platform. The deployment script is run only once, to prepare the Operating System (OS) and install any required dependencies. A snapshot of the prepared OS image is then used for future jobs, when the execution script is called. The execution script validates inputs, executes the application and stores the outputs. In our example, the deployment script creates the appropriate folder structure and installs the required tools: `vina`, `zip`, and `unzip`. After validating the input files, the execution script runs AutoDock Vina with appropriate parameters for each ligand.

4.2.3. Extending Raccoon2 with the RemoteAPI

In order to use the deployed docking tool on a cloud, the WS-PGRADE workflow should be submitted via the gUSE RemoteAPI. A WS-PGRADE workflow consists of an XML file (*workflow.xml*) which describes the workflow, the input files, and contains meta-information, such as which types of cloud instances should be used.

To configure this XML file correctly, the Raccoon2 GUI has been extended to allow users to select the number of cloud instances, their size, the name of the cloud, and the region. As in the original Raccoon2 GUI, the user can attach a set of ligands and a receptor. In the back-end, the attached files are grouped in as many folders as the number of selected cloud instances. On submission, one workflow will be run for each folder.

The updated *workflow.xml* is compressed along with the rest of the input files, following WS-PGRADE naming conventions. Apart from the attached workflow, the RemoteAPI method requires authentication - a password set by the gUSE server administrator, and user credentials. In the current implementation, the end-user is asked to provide these. In line with gUSE conventions, the credentials file should be named `x509._credentialsID` (even if the X.509 standard is not used). The `_credentialsID` should be replaced with the name of the middleware. This file is then compressed and along with the RemoteAPI password and the WS-PGRADE workflow, sent to the gUSE server. The RemoteAPI method returns a *workflowID*.

Monitoring the VS simulation is done by polling the status of the workflow using this *workflowID*. The status is displayed to the user and if there were errors they can re-submit the workflows. Once a workflow has finished, a final RemoteAPI call retrieves the output. When the workflows complete successfully, their output is downloaded and only the relevant AutoDock Vina result files are extracted into a result folder. This folder can be directly used by the *Analysis* tab of Raccoon2 to view the docking results sorted by the AutoDock Vina score. The filtering and visualisation features can be used, exactly as in the original Raccoon2.

4.3. Implementation of Scenario 4

The implementation of Scenario 4 is based on the generic concept described in Section 3.2 and Figure 2. All the elements in the implementation are accessible via a RESTful API implemented using the minimalist web framework "Bottle" [39]. This implementation of Scenario 4 groups the MDRR and DM in a single server (Server 1), and uses a separate server for the ATs for receptor similarity (Server 2), ligand similarity (Server 3), and comparison of configuration files (Server 4), as shown in Figure 4.

If the user has allowed storing the docking results in Raccoon2, they are sent to the MDRR which inserts them in a MongoDB database. The MDRR then sends the target receptor, all other receptors, and a user-provided threshold to the AT DeepAlign on Server 2. Another AT called AssessDeepAlign is responsible for filtering the results of DeepAlign based on the threshold.

Following this, the MDRR selects the ligands and configuration files of previous docking results where the receptor is one of the similar receptors. These ligands will be compared to the target ligand by the AT LIGSIFT. The user-provided threshold value is used by the AT AssessLIGSIFT to determine if two ligands are similar. All configuration files that have been used in a previous docking with a similar ligand and a similar receptor, are sent to the custom-made AT CompareConfig for comparison with the target configuration file. The DM processes the results of all the ATs into one list with all the needed information, ready to be presented to the user.

4.3.1. Implementation of the MDE

The cloud-enabled version of Raccoon2 (as described in Section 4.2) requires some small additions to enable the user to provide the threshold values and select whether the docking results should be stored in the MDRR.

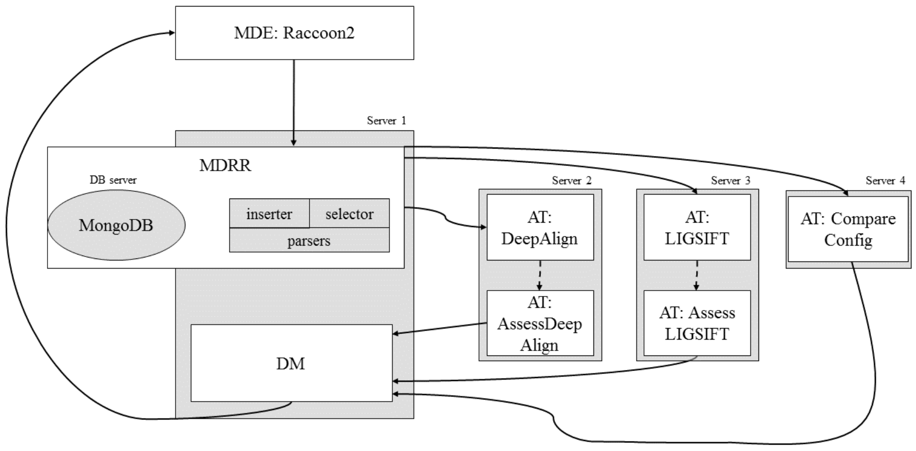


Figure 4: Communication between servers used in the implementation of Scenario 4.

4.3.2. Implementation of the MDRR

This implementation uses PyMongo[40], the Python driver for MongoDB, and PyBel/OpenBabel [41], the libraries for calculating molecular properties. There were several reasons for choosing MongoDB as the underlying database engine:

1. The schema-less design is ideal for polymorphic data.
   (a) Ligand and receptor structures can be stored in a collection regardless of the file format.
   (b) Docking results can be stored in a single collection regardless of the docking tool and file format of the result files.
   (c) One collection can be used for keeping track of all activities (provenance information) regardless of the type of activity, AT or decision made.
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. It is well-suited for prototyping because it is easier to alter what is stored during development.

After considering the feedback from life scientists, a number of molecular properties are calculated and stored alongside the molecular structure. There are a total of 4 collections: *ligands* - ligand structure and properties calculated using PyBel/OpenBabel, *receptors* - receptor structure, *results* - values parsed from the docking result file, and *analysis* - all steps taken to make the final decision.

The MDRR receives three user-input thresholds - for DeepAlign, LIGSIFT, and the custom-made CompareConfig. The MDRR will search the database for previous docking results that have similar input files (receptor, ligand, and docking configuration) to the currently used one.

4.3.3. Implementation of the ATs

AT DeepAlign and AT AssessDeepAlign. In this implementation we have chosen to use DeepAlign, although using several structural alignment tools and combining the results can be beneficial [42]. A new thread, composed of a part that runs DeepAlign and part that compares the DeepScore value from the DeepAlign result to the user-provided threshold, is launched for each receptor pair. If the DeepScore value is greater, the receptors are deemed "similar".

AT LIGSIFT and AT AssessLIGSIFT. Similarly, when comparing ligands, a thread is launched for each pair of ligands. Each thread is composed of a part that runs LIGSFIT and a part that compares the value of ShapeSim to the user-provided threshold. If the threshold is greater than ShapeSim, the ligands are "similar".

AT CompareConfig. An AutoDock Vina configuration file can describe several parameters including the coordinates of a cuboid which should surround the binding site of the receptor. The cuboid is described by two 3-dimensional points, the centre of the cuboid and the "size" of the cuboid (represented by the coordinates of a vertex, relative to the centre). The similarity value we use is the mean of the distances between two 3-dimensional points of the two configuration files:

$$DC = \sqrt{(C_x - OC_x)^2 + (C_y - OC_y)^2 + (C_z - OC_z)^2} \tag{1}$$

$$DS = \sqrt{(S_x - OS_x)^2 + (S_y - OS_y)^2 + (S_z - OS_z)^2} \tag{2}$$

where C_x, C_y, C_z are the coordinates of the centre of a cuboid from a candidate configuration file; OC_x, OC_y, OC_z are the analogous coordinates from the file originally used by the user; S_x, S_y, S_z are the coordinates of the vertex representing the size of the cuboid; and OS_x, OS_y, OS_z are the analogous coordinates from the file originally used by the user.

We call the mean of the two distances, DC and DS, the *comparison value*. If this value is less than the user-provided threshold, then the two cuboids, thus the two configuration files are "similar". The conclusion of this comparison is strictly geometrical, and may not always be biologically correct.

4.3.4. Implementation of the DM

The DM receives the assessed similar receptors, ligands, and configuration files. It loops through all receptors to find similar ligands for that receptor, then for each similar ligand it checks if there were similar configuration files. When these conditions are fulfilled, it adds meta-information such as the similarity results, similarity values, and thresholds. This list is sorted based on the receptor similarity and then based on the ligand similarity value. This is returned to the user, and visualised.

5. Results and Evaluation

5.1. Running VS experiments on multiple heterogeneous clouds from Raccoon2

To test the implementation of the cloud-enabled Raccoon2, we identified biochemically relevant input data. The chosen receptor was *ribokinase* of the protozoan parasite *Trichomonas vaginalis* (TV). TV causes *trichomoniasis*, a very common sexually transmitted infection in humans. A subset of 130,216 ligands have been obtained from the ZINC [43] database of drug-like small molecules. It is a diverse subset of ligands that may bind and antagonise the receptor. We tested the extended Raccoon2 using these input files, conducting three runs of 130,216 docking simulations each. The UoW cloud was used to prove that the approach works, and two runs on the commercial CloudSigma cloud were conducted to show the use of different clouds.

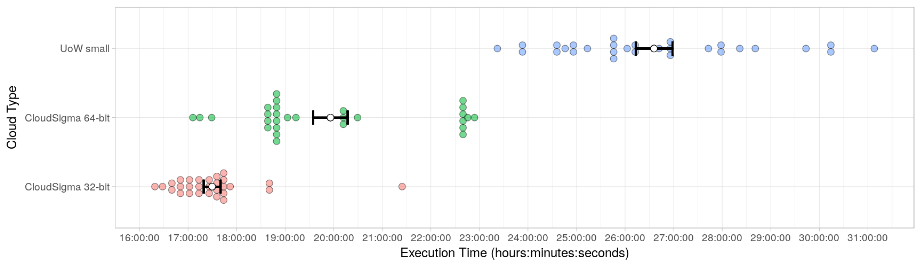


Figure 5: Execution times, mean, and standard error of the mean (x-axis) of the 29 jobs on the three clouds (y-axis).

There are several types of instances that can be used in the UoW cloud: *small* (1-core 2GB RAM), *medium* (2-core 4GB RAM), *large* (4-core 8GB RAM), and *extra-large* (8-core 16GB RAM), all 64-bit. Because this experiment was allocated a maximum capacity of 29 instances and 29 processor cores, we tested our implementation on 29 UoW *small* instances. The mean execution time was *26h 35min 52s* (Figure 5). The CloudSigma cloud had 32-bit or 64-bit CloudSigma *small* (1-core 1GB RAM) instances - note that they have only 1GB RAM. Two experiments were run using these instance types. The mean execution time was *19h 55min 59s* for the 64-bit and *17h 21min 23s* for the 32-bit instances (Figure 5).

AutoDock Vina has been developed for 32-bit processors, but it can be run on 64-bit processors as noted on their official website[4]. However, it seems that the overhead produced is significant and we would recommend using 32-bit cloud instances for this kind of VS experiments since the average execution time decreased by 12.92%. Furthermore, although the CloudSigma instances had half the memory, they finished the docking significantly faster, possibly due to performance optimisations by CloudSigma (on average the 32-bit CloudSigma run was 34.74% faster than the 64-bit UoW run).

[4]`http://vina.scripps.edu/manual.html`

Table 1: Execution times when improving instance type.

Cloud Instances	Mean Execution Time
7 UoW *small*	123h 12min 01s
7 UoW *medium*	75h 35min 16s
7 UoW *large*	51h 47min 29s

Table 2: Execution times when increasing instance number.

Cloud Instances	Mean Execution Time
7 UoW *small*	123h 12min 01s
14 UoW *small*	61h 31min 01s
28 UoW *small*	31h 29min 14s

As of May 2018, CloudSigma's cloud computing prices are \$0.0195 per hour for 1-core CPU, \$0.007 per GB RAM, \$0.1329 per GB SSD storage, and \$0.04 per GB of outbound data transfer [27]. Running the VS experiment on 29 *small* instances would cost \$15.83.

5.1.1. Scalability Tests on the UoW Cloud

In order to show the scalability of this solution we ran several experiments using the same input files described in Section 5.1. Firstly, we ran the VS using the cloud-enabled Raccoon2, selecting 7 *small* instances on the UoW cloud. Then, we increased the instance type to *medium* and finally to *large* while keeping the number of instances to 7. The results in Table 1 show the performance when increasing the number of cores per instance. The left panel of Figure 6 demonstrates the scale-up when compared to an ideal proportional scale-up (where doubling the cores = halving the time).

In a second set of experiments the instance type was kept the same while increasing the number of instances. Namely, the VS was run on 7, 14, and 28 UoW *small* instances (Table 2). The right panel of Figure 6 shows that these results are almost identical to the ideal proportional scale-up. It is clear that although AutoDock Vina has multi-threading capabilities, it is faster to run 28 *small* instances than 7 *large*. Therefore, to maximise efficiency, using more but less powerful instances, rather than less but more powerful instances is recommended.

5.2. Analysing Previous Docking Results with Similar Input

In order to test the implementation of Scenario 4 (Section 4.3), a number of docking simulations were conducted and their results were stored in a sample repository of previous docking results. The extended version of Raccoon2 and the UoW cloud were used to fill this repository.

The repository contains 70 receptors, 7 of them (10%) *a priori* similar to the TV *ribokinase* due to homology, and 63 (90%) other random receptors. A total of 7 structures of other *ribokinase* receptors from 7 different species were

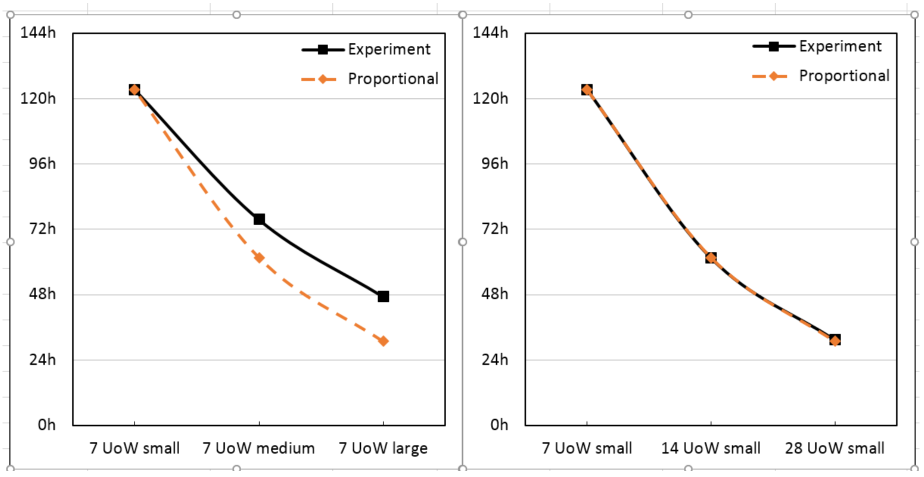


Figure 6: Our experiment compared to a proportional (left, changing instance type, right, changing instance number: y-axis shows execution time in hours, x-axis the cloud instance configuration.

chosen from the PDB [5]. To confirm structural alignment, DeepAlign was run, resulting in high DeepScore values between 975.47 and 1,491.79.

The 63 random receptors were obtained from the RCSB "molecule of the month" series [44] for the months Dec 2012 through Feb 2018. When selecting molecules, the first PDB molecule mentioned in the text was identified, downloaded, converted to .pdbqt, and a test AutoDock Vina docking with a reference ligand was conducted. If any errors appeared in this process, the next PDB molecule mentioned was considered.

Raccoon2 was used to create configuration files for the receptors, with the cuboid covering a part of the receptor which may or may not be biologically important.

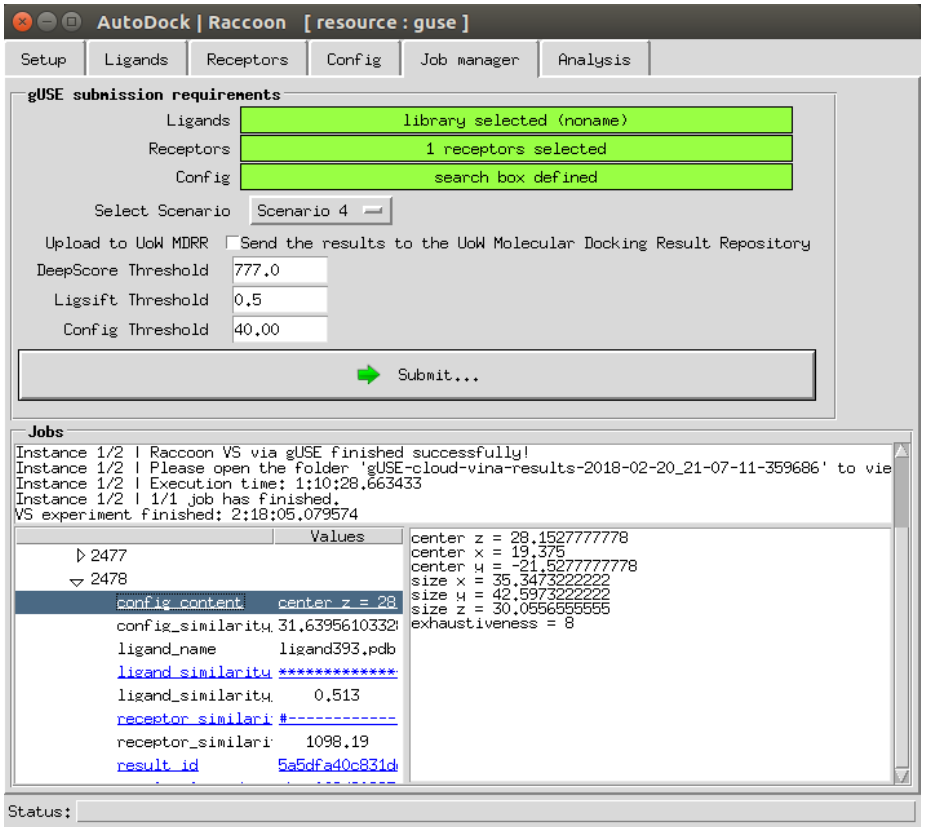
Finally, to obtain a large number of ligands, a set of molecules from ZINC was identified: 2,376 molecules approved as drugs in a jurisdiction in the world [45] were downloaded as one .mol2 file, split and converted to .pdbqt.

The number of docking results stored in the MDRR was 166,320 ($70 \times 2,376$). A total of 80 runs and 393 jobs were executed for this exercise, with a mean execution time of *2h 23min 23s*. This concluded the preparation of the docking repository which was used as the MDRR element for our proof-of-concept of Scenario 4.

To test the proof-of-concept of Scenario 4, a small group of 10 ligands were chosen from the set of 130,216 (Section 5.1) to be docked with the TV *ribokinase* using the extended Raccoon2 as MDE. The threshold values used for this scenario were 777.0 (DeepAlign), 0.5 (LIGSIFT), and 40.0 (CompareConfig).

[5]PDB IDs: 1RKA, 1VM7, 5BYD, 3GO6, 3I3Y, 3RY7, and 4X8F

The four required servers for the MDRR and the ATs were deployed on a local computer with 2.50GHz 4-core CPU and 8GB RAM. The docking on the cloud (2 UoW *small* instances) lasted *07min 21s*, while the entire scenario completed in *2h 18min 5s*. Following this, previous docking results with similar input files can be viewed in the Raccoon2 GUI as shown in Figure 7. The pane on the left-hand side shows the results, and the right-hand side pane shows additional information upon selecting an item.



gUSE submission requirements

Ligands	library selected (noname)
Receptors	1 receptors selected
Config	search box defined
Select Scenario	Scenario 4
Upload to UoW MDRR	Send the results to the UoW Molecular Docking Result Repository
DeepScore Threshold	777.0
Ligsift Threshold	0.5
Config Threshold	40.00

Jobs

```
Instance 1/2 | Raccoon VS via gUSE finished successfully!
Instance 1/2 | Please open the folder 'gUSE-cloud-vina-results-2018-02-20_21-07-11-359686' to vie
Instance 1/2 | Execution time: 1:10:28.663433
Instance 1/2 | 1/1 job has finished.
VS experiment finished: 2:18:05.079574
```

	Values
▷ 2477	
▽ 2478	
config_content	center z = 28
config_similarity	31.6395610332
ligand_name	ligand393.pdb
ligand_similarity	*************
ligand_similarity	0.513
receptor_similari	#------------
receptor_similari	1098.19
result_id	5a5dfa40c831d

```
center z = 28.1527777778
center x = 19.375
center y = -21.5277777778
size x = 35.3473222222
size y = 42.5973222222
size z = 30.0556555555
exhaustiveness = 8
```

Status:

Figure 7: Extended Raccoon2 GUI: Results of Scenario 4.

5.3. *Assessment of data access and data sharing*

If implemented in a large-scale setting, a scenario should provide an efficient way to access and share data. In particular, it should be easy to access the docking results stored in the MDRR by other tools and other scenarios. In our current implementation, the Python web framework Bottle is used to create a minimal working prototype-level RESTful API which is used to transfer data to and from the MDRR. For instance, when the MDE completes the docking experiment, it uses an API method defined within the MDRR to insert the docking results (and corresponding ligands, receptors, and configuration files) into the database. In order to show the results of Scenario 4 to the user (Figure 7), the MDE utilises API methods of the MDRR to retrieve details about a docking result, ligand, receptor, or configuration file, based on their IDs. The communication between the MDRR, ATs, and DM is also done using API calls through HTTP. In this prototype implementation, the API is flexible, but somewhat basic. Additional methods for authentication and detailed data management could be included in future implementations.

In general, the efficiency of data access is particularly important as the size of the data could be a bottleneck and data transfer over the cloud can incur large costs. However, in our example we see that the total size of input and output files of a VS simulation composed of 1 receptor and 130,216 ligands (which produces 130,216 docking results and which we regard as a typical VS simulation) is only 670.1 MB, once compressed. In the current implementation of Scenario 4, all APIs support compressed (.zip) or uncompressed files. The size of the example docking results is relatively small for modern networks and we do not expect this to be a practical bottleneck in terms of bandwidth or time to upload/download. On the other hand, transferring this data on the cloud will incur financial costs which sometimes may be unexpected. Therefore, we recommend having all elements of a scenario and the software that executes the docking on the same cloud. If such an implementation is not feasible, then we suggest using solutions such as the Data Avenue [46] which allows for an efficient transfer of data between different cloud environments. When the MDE is a desktop application that runs the docking simulations on a cloud, like in the example shown in this article, some amount of data has to be transferred between the cloud and the desktop computer. Taking steps, such as compressing the data and transferring only necessary data, minimise the costs due to data transfer.

6. Conclusion and Future Work

This article presented a generic approach to extend domain-specific desktop applications, enabling the execution of simulations on different clouds. By using this approach, existing desktop applications which run on the local machine or expensive compute clusters requiring significant IT support, can be extended to use cloud computing. This makes them more accessible to prospective users who lack the needed funds or expertise to use clusters. Furthermore, unlike existing approaches which require scientists to become familiar with a new custom-made GUI, this approach uses the same familiar GUI of a popular domain-specific application.

We have also explored ways to store and further analyse docking results, easing access to them, and facilitating sharing. While existing systems lack the ability to use the previous docking results of another user, our approach is based around a shared repository of docking results which enables scientists to make conclusions based on the previous results of others.

As future work, we are currently working on the formal description and further generalisation of such molecular docking scenarios. Our aim is to develop a methodology based on this formalism that enables application developers to more efficiently create such extended environments.

Acknowledgments

The research leading to these results has received funding from the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement №608886 (CloudSME) and from the H2020 Programme under Grant Agreement No. 731574 (COLA). The authors would also like to acknowledge funding from the University of Westminster Research Studentship 2014.

References

[1] Z. Xiang, Advances in homology protein structure modeling, Curr Protein Pept Sci 7 (3) (2006) 217–227. `doi:10.2174/138920306777452312`.

[2] P. Mell, T. Grance, The NIST definition of cloud computing, Commun. ACM 53 (6) (2010) 50.

[3] K. Wolstencroft, R. Haines, D. Fellows, A. Williams, D. Withers, S. Owen, S. Soiland-Reyes, I. Dunlop, A. Nenadic, P. Fisher, J. Bhagat, K. Belhajjame, F. Bacall, A. Hardisty, A. N. de la Hidalga, M. P. B. Vargas, S. Sufi, C. Goble, The Taverna workflow suite: designing and executing workflows of web services on the desktop, web or in the cloud, Nucleic Acids Res. 41 (W1) (2013) W557–W561. `doi:10.1093/nar/gkt328`.

[4] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, Y. Zhao, Scientific workflow management and the Kepler system, Concurrency Computat Pract Exper. 18 (10) (2006) 1039–1065. `doi:10.1002/cpe.994`.

[5] P. Kacsuk, K. Karoczkai, G. Hermann, G. Sipos, J. Kovacs, WS-PGRADE: Supporting parameter sweep applications in workflows, in: 2008 Third Workshop on Workflows in Support of Large-Scale Science, IEEE, 2008, pp. 1–10. `doi:WORKS.2008.4723955`.

[6] M. M. Jaghoori, A. J. Altena, B. Bleijlevens, S. Ramezani, J. L. Font, S. D. Olabarriaga, A multi-infrastructure gateway for virtual drug screening, Concurrency Computat Pract Exper. 27 (16) (2015) 4478–4490. `doi:10.1002/cpe.3498`.

[7] J. Krüger, R. Grunzke, S. Gesing, S. Breuers, A. Brinkmann, L. de la Garza, O. Kohlbacher, M. Kruse, W. E. Nagel, L. Packschies, R. Müller-Pfefferkorn, P. Schäfer, C. Schärfe, T. Steinke, T. Schlemmer, K. D. Warzecha, A. Zink, S. Herres-Pawlis, The MoSGrid science gateway–a complete solution for molecular simulations, J. Chem. Theory Comput. 10 (6) (2014) 2232–2245. `doi:10.1021/ct500159h`.

[8] T. Kiss, P. Greenwell, H. Heindl, G. Terstyanszky, N. Weingarten, Parameter sweep workflows for modelling carbohydrate recognition, J. Grid Comput. 8 (4) (2010) 587–601. doi:10.1007/s10723-010-9166-8.

[9] S. Forli, R. Huey, M. E. Pique, M. F. Sanner, D. S. Goodsell, A. J. Olson, Computational protein–ligand docking and virtual drug screening with the autodock suite, Nat. Protoc. 11 (5) (2016) 905. doi:10.1038/nprot.2016.051.

[10] P. D'Ursi, F. Chiappori, I. Merelli, P. Cozzi, E. Rovida, L. Milanesi, Virtual screening pipeline and ligand modelling for H5N1 neuraminidase, Biochem. Biophys. Res. Commun. 383 (4) (2009) 445–449. doi:10.1016/j.bbrc.2009.04.030.

[11] D. Temelkovski, T. Kiss, G. Terstyanszky, Molecular docking with Raccoon2 on clouds: extending desktop applications with cloud computing, in: Proceedings of the 9th International Workshop on Science Gateways (IWSG 2017), 2017, to be published.

[12] R. De Paris, F. A. Frantz, O. Norberto de Souza, D. D. Ruiz, wFReDoW: A cloud-based web environment to handle molecular docking simulations of a fully flexible receptor model, BioMed Res. Int. 2013. doi:10.1155/2013/469363.

[13] G. M. Morris, R. Huey, W. Lindstrom, M. F. Sanner, R. K. Belew, D. S. Goodsell, A. J. Olson, AutoDock4 and AutoDockTools4: Automated docking with selective receptor flexibility, J Comput Chem 30 (16) (2009) 2785–2791.

[14] Amazon Web Services, Amazon EC2, https://aws.amazon.com/ec2/, Online; Accessed 10 May 2018.

[15] H. Berman, K. Henrick, H. Nakamura, J. L. Markley, The worldwide Protein Data Bank (wwPDB): ensuring a single, uniform archive of PDB data, Nucleic Acids Res. 35 (suppl_1) (2006) D301–D303. doi:doi.org/10.1093/nar/gkl971.

[16] S. R. Ellingson, J. Baudry, High-throughput virtual molecular docking with AutoDockCloud, Concurrency Computat Pract Exper. 26 (4) (2014) 907–916. doi:doi.org/10.1002/cpe.2926.

[17] N. Huang, B. K. Shoichet, J. J. Irwin, Benchmarking sets for molecular docking, J. Med. Chem. 49 (23) (2006) 6789–6801. doi:10.1021/jm0608356.

[18] T. Kiss, P. Borsody, G. Terstyanszky, S. Winter, P. Greenwell, S. McEldowney, H. Heindl, Large-scale virtual screening experiments on Windows Azure-based cloud resources, Concurrency Computat Pract Exper. 26 (10) (2014) 1760–1770. doi:10.1002/cpe.3113.

[19] O. Trott, A. J. Olson, AutoDock Vina: Improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading, J. Comput. Chem. 31 (2) (2010) 455–461. doi:10.1002/jcc.21334.

[20] Z. Farkas, P. Kacsuk, T. Kiss, P. Borsody, Á. Hajnal, Á. Balaskó, K. Karóczkai, AutoDock gateway for molecular docking simulations in cloud systems, in: O. Terzo, M. Lorenzo (Eds.), Cloud Computing with E-science Applications, Vol. 1, Boca Raton, Florida : CRC Press, 2015, Ch. 10, pp. 217–237.

[21] I. Sánchez-Linares, H. Pérez-Sánchez, J. M. Cecilia, J. M. García, High-throughput parallel blind virtual screening using BINDSURF, BMC Bioinformatics 13 (14) (2012) S13. doi:10.1186/1471-2105-13-S14-S13.

[22] N. D. Prakhov, A. L. Chernorudskiy, M. R. Gainullin, VSDocker: a tool for parallel high-throughput virtual screening using AutoDock on Windows-based computer clusters, Bioinformatics 26 (10) (2010) 1374–1375. doi:10.1093/bioinformatics/btq149.

[23] X. Jiang, K. Kumar, X. Hu, A. Wallqvist, J. Reifman, DOVIS 2.0: an efficient and easy to use parallel virtual screening tool based on AutoDock 4.0, Chem. Cent. J. 2 (1) (2008) 18. doi:10.1186/1752-153X-2-18.

[24] X. Zhang, S. E. Wong, 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 54 (1) (2014) 324–337. doi:10.1021/ci4005145.

[25] L. Xie, T. Evangelidis, L. Xie, P. E. Bourne, Drug discovery using chemical systems biology: Weak inhibition of multiple kinases may contribute to the anti-cancer effect of Nelfinavir, PLOS Comput. Biol 7 (4) (2011) e1002037. doi:10.1371/journal.pcbi.1002037.

[26] E. Glaab, Building a virtual ligand screening pipeline using free software: a survey, Brief. Bioinform 17 (2) (2015) 352–366. doi:10.1093/bib/bbv037.

[27] Cloudsigma Holding AG, Cloud servers & Hosting, https://www.cloudsigma.com, Online; Accessed 10 May 2018.

[28] S. Wang, J. Ma, J. Peng, J. Xu, Protein structure alignment beyond spatial proximity, Sci. Rep. 3 (2013) 1448. doi:10.1038/srep01448.

[29] A. Roy, J. Skolnick, LIGSIFT: an open-source tool for ligand structural alignment and virtual screening, Bioinformatics 31 (4) (2014) 539–544. doi:10.1093/bioinformatics/btu692.

[30] Z. Vincent, D. Antoine, Click2Drug: Directory of in silico drug design tools, `http://www.click2drug.org/index.html#Docking`, online; Accessed 22 Nov 2018 (Sep 2017).

[31] W. J. Allen, T. E. Balius, S. Mukherjee, S. R. Brozell, D. T. Moustakas, P. T. Lang, D. A. Case, I. D. Kuntz, R. C. Rizzo, DOCK 6: impact of new features and current docking performance, J. Comput. Chem. 36 (15) (2015) 1132–1156.

[32] M. L. Verdonk, J. C. Cole, M. J. Hartshorn, C. W. Murray, R. D. Taylor, Improved protein–ligand docking using GOLD, Proteins: Structure, Function, and Bioinformatics 52 (4) (2003) 609–623.

[33] B. Kramer, M. Rarey, T. Lengauer, Evaluation of the FlexX incremental construction algorithm for protein–ligand docking, Proteins: Structure, Function, and Bioinformatics 37 (2) (1999) 228–241.

[34] S. F. Sousa, P. A. Fernandes, M. J. Ramos, Protein-ligand docking: Current status and future challenges, Proteins: Structure, Function, and Bioinformatics 65 (1) (2006) 15–26, 00513. `doi:10.1002/prot.21082`.

[35] P. Kacsuk, Z. Farkas, M. Kozlovszky, G. Hermann, A. Balasko, K. Karoczkai, I. Marton, WS-PGRADE/gUSE generic DCI gateway framework for a large variety of user communities, J. Grid Comput. 10 (4) (2012) 601–630. `doi:10.1007/s10723-012-9240-5`.

[36] M. Kozlovszky, K. Karóczkai, I. Márton, P. Kacsuk, T. Gottdank, DCI bridge: Executing WS-PGRADE workflows in distributed computing infrastructures, in: Science Gateways for Distributed Computing Infrastructures, Springer, 2014, pp. 51–67. `doi:10.1007/978-3-319-11268-8_4`.

[37] S. J. Taylor, T. Kiss, G. Terstyanszky, P. Kacsuk, N. Fantini, Cloud computing for simulation in manufacturing and engineering: introducing the CloudSME simulation platform, in: Proceedings of the 2014 Annual Simulation Symposium, Society for Computer Simulation International, 2014, p. 12.

[38] C. GmbH., CloudBroker Platform, `http://cloudbroker.com/platform/`, Online; Accessed 10 May 2018.

[39] M. Hellkamp, Bottle: Python Web Framework - Bottle 0.13-dev documentation, `https://bottlepy.org/docs/dev/`, Online; Accessed 10 May 2018.

[40] M. Dirolf (mdirolf), et al., PyMongo 3.6.1 Documentation - PyMongo 3.6.1 documentation, `https://api.mongodb.com/python/current/`, Online; Accessed 10 May 2018.

[41] N. M. O'Boyle, C. Morley, G. R. Hutchison, Pybel: a Python wrapper for the OpenBabel cheminformatics toolkit, Chem. Cent. J. 2 (1) (2008) 5. `doi:10.1186/1752-153X-2-5`.

[42] R. Kolodny, P. Koehl, M. Levitt, Comprehensive evaluation of protein structure alignment methods: Scoring by geometric measures, J. Mol. Biol. 346 (4) (2005) 1173–1188. `doi:10.1016/j.jmb.2004.12.032`.

[43] J. J. Irwin, B. K. Shoichet, Zinc–a free database of commercially available compounds for virtual screening, J. Chem. Inf. Model. 45 (1) (2005) 177–182. `doi:10.1021/ci049714+`.

[44] D. S. Goodsell, S. Dutta, C. Zardecki, M. Voigt, H. M. Berman, S. K. Burley, The RCSB PDB "molecule of the month": Inspiring a molecular view of biology, PLoS biology 13 (5) (2015) e1002140. `doi:10.1371/journal.pbio.1002140`.

[45] ZINC15, Subset of approved molecules approved in major jurisdictions including the FDA, `http://zinc15.docking.org/substances/subsets/world/`, Online; Accessed 10 May 2018.

[46] Á. Hajnal, Z. Farkas, P. Kacsuk, Data avenue: Remote storage resource management in WS-PGRADE/gUSE, in: Proceedings of the 6th International Workshop on Science Gateways (IWSG 2014), IEEE, 2014, pp. 1–5.