Molecular docking with Raccoon2 on clouds: extending desktop applications with cloud computing

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 is a computer simulation that predicts the binding affinity between two molecules, a ligand and a receptor. Large-scale docking simulations, using one receptor and many ligands, are known as structure-based virtual screening. Often used in drug discovery, virtual screening 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. Cloud computing provides on-demand availability, pay-per-use pricing, and great scalability which can improve the performance and efficiency of scientific applications. This paper investigates how domain-specific desktop applications can be extended to run scientific simulations on various clouds. A generic approach based on scientific workflows is proposed, and a proof of concept is implemented using the Raccoon2 desktop application for virtual screening, WS-PGRADE workflows, and gUSE services with the CloudBroker platform. The presented analysis illustrates that this approach of extending a domain-specific desktop application can run workflows on different types of clouds, and indeed makes use of the on-demand scalability provided by cloud computing. It also facilitates 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.**

Keywords—cloud computing; molecular docking; Raccoon2; virtual screening; WS-PGRADE/gUSE; bioinformatics

I. 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 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 determined using X-ray crystallography or NMR spectroscopy, or estimated using homology modelling. Molecular docking consists of an algorithm to search through the conformational space of the molecules, and a scoring function to estimate the energy between the ligand and the receptor's binding site. 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 high throughput screening, the automated laboratory experiment). In practice, as in the rest of this paper, these terms are often 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).

Cloud computing is a paradigm based on virtualisation of data centres, 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" [1]. The scalability and elasticity provided by cloud computing makes it useful for VS. Clouds are available on-demand and users are charged on pay-per-use basis. This can make scientific applications, such as VS, more accessible for life scientists around the world, lowering the cost of using complex computing infrastructure. If VS is implemented based on the Software-as-a-Service model, life scientists will always have access to the latest version of the simulation software. Scientists and students without access to expensive DCIs, and without experience in configuring them, will be able to run VS easily.

Scientific workflow systems such as Taverna [2], Kepler [3] or WS-PGRADE [4], provide a convenient way to represent and develop complex applications composed of multiple steps and executables. A user-friendly interface is usually used to provide 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 [5]-[7]. 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 [8]. 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.

This paper describes 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. We demonstrate the utilisation of

This work was funded by the CloudSME Cloud-Based Simulation platform for Manufacturing and Engineering Project No. 608886 and the COLA Cloud Orchestration at the level of Applications Project No. 731574 projects.

CEUR-WS.org/Vol-2363/paper2.pdf

distributed heterogeneous clouds to run VS, noting that various other DCIs can also be used with little or no changes to our approach.

II. Related Work

Most VS experiments on DCIs have used CPU, GPU clusters or grid computing resources (e.g. [5], [6], [9]-[11]). Applying cloud computing for such experiments is still relatively new with much lower number of examples.

One such example is the wFReDoW [12], a web-based environment for docking flexible receptors using the docking tool AutoDock 4.2. Their infrastructure is comprised of a virtual MPI master-slave environment deployed on the commercial Amazon EC2 cloud [13]. They have tested it using five c1.xlarge EC2 Amazon instances (8-cores, 7GB RAM, 1.65TB storage), the ligand *triclosan* obtained from the PDB (Protein Data Bank) [14] (PDB ID: 1P45A), and 3,100 snapshots of the receptor *InhA* from *Mycobacterium tuberculosis*, generated by Molecular Dynamics (MD) simulations to model the flexibility.

Another example is AutoDockCloud [15] which uses Hadoop and MapReduce to run AutoDock4 on the private "Kandinsky" cloud at the Oak Ridge National Laboratory. With 57 16-core nodes reserved for MapReduce, 570 docking simulations can be performed simultaneously. To test their solution, the authors used the human *estrogen receptor alpha* obtained from the PDB (PDB ID: 1L2I) and 2,637 ligands from the DUD [16] database. AutoDockCloud has completed the VS 450 times faster than a non-parallel execution without affecting the biochemical results. However, it only handles the docking stage and not the pre or post-docking preparation and analysis, which have to be done separately.

In a third example [17], AutoDock and AutoDock Vina have been ported on the Windows Azure-based "VENUS-C" cloud computing service. Using a small desktop application, scientists were able to submit, monitor and retrieve results of VS simulations. 10,000 ligands and a receptor, generated from a short MD run on an initial structure, have been tested on 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. While these experiments illustrated the applicability of cloud computing resources, the custom user interface was rather simplistic restricting the depth of experiments.

As summarised above, there have been several efforts to run VS using cloud computing. However, all these attempts provided their own restricted GUI for running the simulations and were focusing on a specific cloud computing infrastructure. In comparison, the approach suggested in this paper, enables scientists to use the GUI of a popular domain-specific desktop application they are familiar with. Furthermore, Raccoon2, the desktop application we use to prove our concept, provides pre-docking and post-docking facilities to prepare input files, and analyse results. Finally, our approach utilises a set of services in the form of WS-PGRADE/gUSE and the CloudBroker Platform which support a wide range of clouds as well as other DCIs, not limiting life scientists to using a specific infrastructure.

III. Generic Concept

Our aim is to enable existing desktop applications to access heterogeneous cloud computing resources. This should be achieved without major reengineering of the desktop application and without further burdening the end-user. Ideally, 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 objective, a set of services (we name them Cloud Access Services - CAS) can be called from the desktop application. CAS should be available from an Application Programming Interface (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, and should enable the design and execution of complex application scenarios, such as parameter sweep workflow applications, typically required to design VS experiments.

The integration requires two major steps from the developers, as illustrated in Fig. 1. During the first step, CAS is configured to run the application in the cloud. This step typically requires preparing workflow applications describing the experiment, and configuring CAS to interface with the desired cloud resources. In the second step minor modification of the GUI of the desktop application is required while integrating the execution of the workflow (practically creating a simple button to execute the complex workflow representing the experiment), and to retrieve the results. Instead of implementing CAS, the core component of this conceptual architecture from scratch, existing tools to support the creation of parameter sweep workflows 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. The rest of this paper describes this approach using a set of existing services and components as the selected CAS and their integration to a VS desktop application.

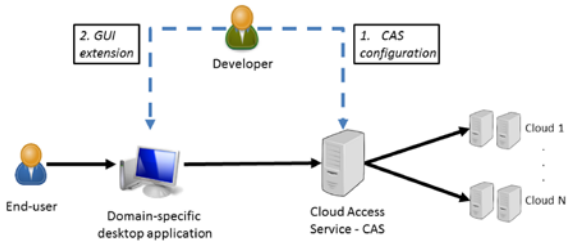

Fig. 1. Generic concept for extending desktop applications to run on clouds

IV. Background

The solution developed and presented in this paper is focused around the extension of the VS tool Raccoon2, by connecting it to a WS-PGRADE/gUSE science gateway and to various cloud computing resources via the CloudBroker

Platform. This section briefly describes these technologies and components. When connected, they enable scientists to run VS simulations on various clouds using a familiar GUI.

A. *Raccoon2 and AutoDock Vina*

Raccoon2 is the latest version of an open-source desktop application for preparing and analysing VS with AutoDock Vina. It supports executing docking experiments a Linux cluster with the PBS or SGE schedulers, and it incorporates analysis features such as filtering, visualising, and exporting the results. Users can select ligands and receptors, configure docking options, visualise a binding site, and connect to a cluster to submit jobs, directly from the Raccoon2 GUI. We have chosen Raccoon2 in our implementation since it is the VS tool of choice for bio-scientists at the University of Westminster (UoW).

Before the jobs can be submitted, Raccoon2 guides users to deploy the docking tool AutoDock Vina on a cluster. AutoDock Vina [18] is an open-source docking tool with built-in support for multithreading. It uses a hybrid global-local conformation search with a gradient-based local optimisation, and 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.

B. *WS-PGRADE/gUSE*

WS-PGRADE/gUSE [19] is a workflow-centric open-source science gateway framework. WS-PGRADE workflows are dataflow directed acyclic graphs where nodes represent execution blocks, with input and output ports, which can be executed in parallel. 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 a Liferay-based [20] e-science web portal for development of parallel applications executed on various DCIs using WS-PGRADE workflows. It has a graph editor which allows creating, configuring and executing workflows using gUSE (Grid and cloud User Support Environment) services. The gUSE is an open-source service stack that can form the back-end of science gateways executing applications on DCIs. It provides well-defined services for workflow management. Originally supporting primarily service grids, desktop grids, and clusters, gUSE also supports parallel execution on clouds. A gUSE internal component called DCI Bridge [21], provides a well-defined communication interface enabling access to many different DCIs, including clouds [22].

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

C. *CloudBroker Platform*

The CloudBroker Platform [23] 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 kinds of clouds, including commercial (e.g. CloudSigma, Amazon Web Services) and open-source (e.g. OpenNebula, OpenStack). The CloudBroker platform has been integrated into gUSE's DCI Bridge, providing various cloud computing resources to the WS-PGRADE/gUSE framework.

V. Design and Implementation

Based on the generic concept described in Section III and the tools introduced in Section IV, a reference implementation of the proposed architecture has been completed. When implementing the generic concept of Fig. 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 infrastructures are the UoW OpenStack cloud, and the CloudSigma [25] cloud (Fig. 2).

As described in Section III, the development is divided into 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 well-configured WS-PGRADE workflow needs to be attached. To simplify this step, a developer can create the workflow using a WS-PGRADE portal, test it with test input data, and then export it. The exported workflow can be configured from the code of the domain-specific desktop application and attached to a RemoteAPI call, rather than created from scratch. To conclude (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 the domain-specific desktop application is extended, in order to make the appropriate RemoteAPI calls. The next sections will elaborate on these steps.

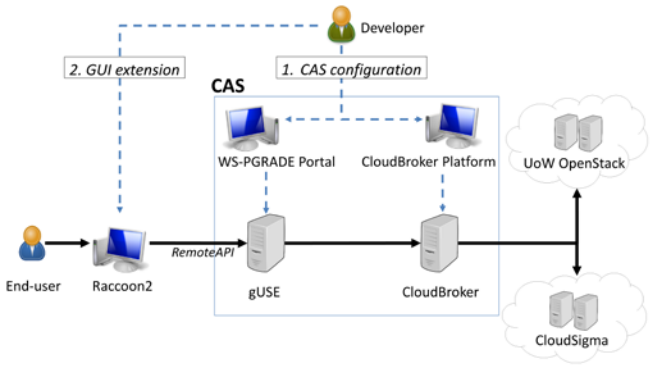

Fig. 2. Architecture of our reference implementation using Raccoon2, WS-PGRADE/gUSE, CloudBroker, and the UoW or CloudSigma clouds

A. Creating the WS-PGRADE Workflow

The execution steps of the domain-specific desktop application are recreated using a WS-PGRADE workflow. In this particular case a simple one-node workflow with four input (ligand files, receptor file, Vina configuration file, and an additional file to overcome an output names issue) and one output (the zipped results from the multiple docking runs) ports were created. However, please note that based on the desktop application more complex workflows may be required.

After consultation with life scientists, an issue with previous implementations of AutoDock Vina as a WS-PGRADE workflow was discovered. It was difficult to understand which output file resulted from which input due to automatically generated output names. AutoDock Vina, by default, uses the names of the input ligand and receptor to name the result file. However, WS-PGRADE changes the names of parametric input files internally. In order to avoid this, we do not to use parametric input ports, but rather apply an additional text file which contains pre-generated names of the expected result file, for each ligand-protein pair. This file is created within Raccoon2 before the workflow is submitted.

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, our extended code of Raccoon2 splits the set of ligands into as many zip archives as the number of instances, and submits a separate workflow to each instance.

B. 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. If an image of the Operating System (OS) is not present in the image repository of the target cloud, it should also be installed. The deployment script is run only once, to prepare the 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 a particular folder.

In our example extension of Raccoon2, Ubuntu 14.04 is used to run the deployment script which creates the appropriate folder structure and installs the required tools: AutoDock Vina, zip, and unzip. After validating the input files, the execution script runs AutoDock Vina with appropriate parameters for each ligand. This deployment process is simplified because AutoDock Vina is included in the Ubuntu package repository.

C. Extending the Raccoon2 Source Code with the RemoteAPI

In order to conduct VS on a cloud, we need to submit the WS-PGRADE workflow using the gUSE RemoteAPI. A WS-PGRADE workflow consists of an XML file (*workflow.xml*) which describes the workflow and the input files. The XML file contains other valuable information, such as which kinds of cloud instances would be used.

To fill in the cloud configuration information correctly, we added a section in the Raccoon2 GUI which enables users to select the number of cloud instances, their size, the name of the cloud, and the region. We store all possible values, and their encoded versions, in an additional XML file (*gUSECloudConfiguration.xml*). At the moment, this file is manually synchronised to contain the correct values of all supported types of cloud instances. If, for example, there is a change in the maximum number of instances a cloud can handle, a value in this file should be changed.

Within the original Raccoon2 GUI, the user can attach a set of ligands and a receptor. In our extension, 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, where its results will be stored after downloading. Before submitting it, the WS-PGRADE workflow file is configured to include the cloud selected in the GUI by the scientist.

Finally the updated *workflow.xml* file is zipped along with the rest of the input files, following the WS-PGRADE naming convention, to compose a well-formed WS-PGRADE workflow. Thus, the Raccoon2 code can submit it by calling a gUSE RemoteAPI method using the command *curl*.

Apart from the attached workflow, this RemoteAPI method requires authentication. Namely, it needs a RemoteAPI password set by the gUSE server administrators, and CloudBroker user credentials (username and password). In the current implementation, for security reasons, the end-user is asked to provide these. In line with gUSE conventions, the credentials file should be named *x509._credentialsID*, where *_credentialsID* is the name of the middleware that requires the authentication (this naming convention remains, even though the X.509 standard is not used). This file is then zipped and along with the RemoteAPI password and the zipped WS-PGRADE workflow, they are sent as POST parameters. The RemoteAPI method returns a *workflowID*, which is used to check the workflow's status.

Monitoring the VS simulation is done by polling for the status of the workflows using the RemoteAPI. In our current implementation, a status check is performed every 20 seconds. 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 (*.pdbqt_log.txt* and *.pdbqt*) are extracted into a result folder. This folder can be directly used by the original analysis tab of Raccoon2. The scientist needs to simply select it in order to view the ligands sorted by the docking results. The filtering and visualisation features can be used, exactly as in the original Raccoon2. The source code of our extended version of Raccoon2 is available at https://github.com/damjanmk/Raccoon2.

VI. Results and Evaluation

A. Proof of Concept on the UoW and CloudSigma Clouds

To show that our concept can be implemented to run a real-life VS on different clouds, we obtained biochemically relevant

input data from life scientists. The receptor is an enzyme called *ribokinase*, which is part of the salvage pathway of nucleotides in the protozoan parasite *Trichomonas Vaginalis (TV)*. The 3D structure of this receptor has been created by homology modelling. TV causes trichomoniasis, a very common sexually transmitted infection. A set of 130,216 ligands have been obtained from the ZINC [24] database of drug-like small molecules. It is a diverse subset of ligands that may bind and antagonise the receptor. We tested our extended Raccoon2 using these input files, conducting three runs, effectively 130,216 docking simulations each. The UoW OpenStack cloud (at London, UK) was used to prove that the approach works, and two runs on the commercial CloudSigma cloud (at Zürich, Switzerland) were conducted to show the use of different clouds.

There are several types of 64-bit (x86_64) 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). At the time of the tests, the UoW cloud had a maximum capacity of 29 instances and processor cores that could be allocated for this experiment. Therefore, we tested our implementation on 29 *small* instances. In order to do this the extended code of Raccoon2 split the 130,216 ligands into 29 groups. A total of 7 jobs (24.14%) had errors due to connection problems between CloudBroker and UoW cloud, but all finished successfully after re-submission. The average execution time per instance was **26h 35min 52s**. To compare the results of both clouds, we decided to use 29 instances most similar in type to the UoW *small* instances.

There are 32-bit or 64-bit CloudSigma small (1-core 1GB RAM) instances, note that they have only 1GB RAM. There were noticeable differences in the execution time between the fastest and the slowest job per run (e.g. 64-bit fastest: 17h 5min 57s; slowest: 22h 53min 59s). The average time per job for the 64-bit instances was **19h 55min 59s**, while the 32-bit instances, **17h 21min 23s**. Fig. 3 shows the execution time for each of the 29 jobs (instances numbered 1, 2, 3, etc. in each run may be docking different ligands).

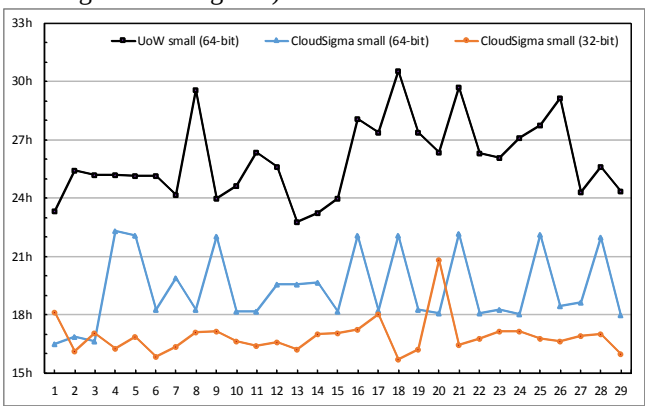

Fig. 3. Comparison of the execution times of each of the 29 instances

The AutoDock Vina software has been developed for 32-bit machines and as noted on their official website, it is compatible with 64-bit machines (http://vina.scripps.edu/manual.html). However, it seems that the overhead produced is significant and we can generally 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, due to various performance optimisations in the CloudSigma cloud, they finished the docking significantly faster (on average the 32-bit CloudSigma run was 34.74% faster than the 64-bit UoW run).

B. *Scalability tests on the UoW cloud*

In order to show the scalability of our solution we designed several more experiments using the same input files described in part A. Firstly, we ran the VS using our cloud-enabled Raccoon2, selecting 7 *small* instances on the UoW cloud. The average time per instance was **123h 12min 1s**. Then, we increased the instance type to *medium* while keeping the number of instances to 7. The average time per instance was **75h 35min 16s**. Finally, we used 7 *large* instances, resulting in average time per instance of **51h 47min 29s**. These results demonstrate reasonable scalability of Raccoon2 when increasing the number of cores inside the instances. The left panel of Fig. 4 demonstrates the scale-up when compared to an ideal proportional scale-up (double the cores = half the time).

In a second set of experiments we kept the instance type the same (UoW small) while increasing the number of instances. Namely, we ran 14 *small* instances with the average time per instance of **61h 31min 1s**, followed by 28 small instances resulting with average time per instance of **31h 29min 14s**. The right panel of Fig. 4 shows that these results very closely resemble the ideal proportional scale-up. It shows that although AutoDock Vina has multithreading capabilities, it is faster to run 28 small instances than 7 large. Therefore, to maximise efficiency, we can recommend using more, but less powerful, rather than less, but more powerful instances.

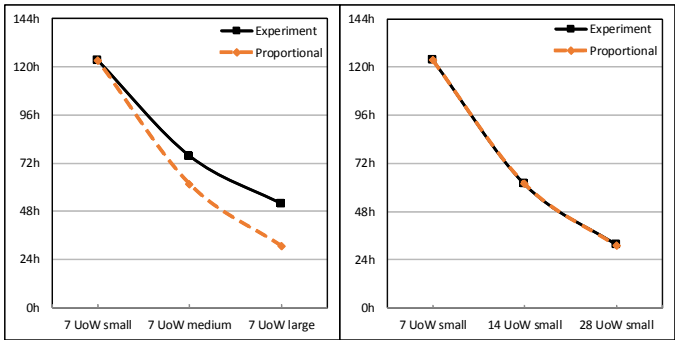

Fig. 4. Scalability - Our experiment compared to a proportional: increasing the configuration of instances (left), increasing the number of instances (right)

C. *Price Estimate*

As of March 2017, CloudSigma 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 [25]. Therefore, running our VS on 29 *small* instances would cost *$15.83*.

D. *Exploring the potential of using other DCIs*

As WS-PGRADE/gUSE is connected to other DCIs such as desktop grids, clusters or service grids via the DCI Bridge, the

same generic solution and the same workflow mapped to these different resources can also be applied to further extend the applicable resources of the experiments. In order to demonstrate this possibility the experiments were executed on the SZTAKI Desktop Grid (SZDG), a BOINC-based desktop grid, integrated in gUSE's DCI Bridge [26]. Desktop grids use spare CPU cycles from desktop computers to create a powerful DCI. To prove our concept, we used a WS-PGRADE portal (https://autodock-portal.sztaki.hu/liferay-portal-6.1.0) to run AutoDock Vina workflows on the SZDG using the same input as above 5 times, with average execution time of 30h 16min 9s.

VII. Conclusion and Future Work

This paper presented a generic approach to extend domain-specific desktop applications, enabling the execution of simulations on different clouds. Several experiments were run to test and evaluate our approach on two different cloud infrastructures and measure the scalability of our solution. We noticed better performance when using many smaller rather than a few larger, and 32-bit rather than 64-bit instances. Although our implementation is based on the VS software Raccoon2, WS-PGRADE/gUSE and CloudBroker, the concept of extending desktop applications to run on clouds is generic. With our extension, Raccoon2 users can use the same familiar GUI to run their VS experiments on clouds. They no longer require access to a Linux PBS CPU or GPU cluster, which brings down the cost of running large VS simulations, making them affordable for scientists around the world. As shown in our tests, the solution works for different kinds of clouds. Considering all gUSE supported DCIs, it could use clusters, grids, and desktop grids.

In the future, we plan to test our implementation on other DCIs and different desktop applications. Instead of the classical method used here, container technology could help automate the software deployment. At the moment, due to the nature of the gUSE RemoteAPI, result files can only be downloaded to the user's desktop. We will focus future work on exploring ways to store and further analyse them, easing access to, and facilitating the sharing of docking results. This can provide a training set for machine learning-based prediction of execution time and cost, and optimising cloud resource utilisation.

References

[1] P. Mell, T. Grance, and others, "The NIST definition of cloud computing", 2011.

[2] K. Wolstencroft et al., "The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud", *Nucleic Acids Res.*, vol. 41, no. W1, pp. W557-W561, Jul, 2013.

[3] B. Ludäscher et al., "Scientific workflow management and the Kepler system", *Concurr. Comp. Pract E.*, vol. 18, no. 10, pp. 1039-1065, Aug, 2006.

[4] P. Kacsuk, K. Karoczkai, G. Hermann, G. Sipos, and J. Kovacs, "WS-PGRADE: supporting parameter sweep applications in workflows", in The 3rd workshop on Workflows in Support of Large-Scale Science, WORKS 2008, Austin, TX, USA, 17 Nov 2008, IEEE, 2008. pp. 1-10.

[5] M. M. Jaghoori, A. J. van Altena, B. Bleijlevens, S. Ramezani, J. L. Font, and S. D. Olabarriaga, "A multi-infrastructure gateway for virtual drug screening", *Concurr. Comp. Pract E.*, vol. 27, no. 16, pp. 4478-4490, Nov, 2015.

[6] J. Krüger et al., "Performance studies on distributed virtual screening", *BioMed Res. Int.*, vol. 2014, pp. 1-7, Jun, 2014.

[7] T. Kiss, P. Greenwell, H. Heindl, G. Terstyanszky, and N. Weingarten, "Parameter sweep workflows for modelling carbohydrate recognition", *J. Grid Comput.*, vol. 8, no. 4, pp. 587-601, Dec, 2010.

[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] N. D. Prakhov, A. L. Chernorudskiy, and M. R. Gainullin, "VSDocker: a tool for parallel high-throughput virtual screening using AutoDock on Windows-based computer clusters", *Bioinformatics*, vol. 26, no. 10, pp. 1374-1375, May, 2010.

[10] X. Jiang, K. Kumar, X. Hu, A. Wallqvist, and J. Reifman, "DOVIS 2.0: an efficient and easy to use parallel virtual screening tool based on AutoDock 4.0", *Chem. Cent. J.*, vol. 2, no. 1, p. 18, Sep, 2008.

[11] I. Sánchez-Linares, H. Pérez-Sánchez, J. Cecilia, and J García, "High-Throughput parallel blind Virtual Screening using BINDSURF", *BMC Bioinformatics*, vol. 13, suppl. 14, S13, Sep, 2012.

[12] R. De Paris, F. A. Frantz, O. Norberto de Souza, and D. D. A. Ruiz, "wFReDoW: a cloud-based web environment to handle molecular docking simulations of a fully flexible receptor model", *BioMed Res. Int.*, vol. 2013, pp. 1-12, Mar, 2013.

[13] Amazon Web Services, Inc. "Amazon EC2". [Online]. Available: https://aws.amazon.com/ec2/. [Accessed: 7 Mar 2017]

[14] H. Berman, K. Henrick, H. Nakamura, and J. L. Markley, "The worldwide Protein Data Bank (wwPDB): ensuring a single, uniform archive of PDB data", *Nucleic Acids Res.*, vol. 35, Database Issue, pp. D301-D303, Jan, 2007.

[15] S. R. Ellingson and J. Baudry, "High-throughput virtual molecular docking with AutoDockCloud", *Concurr. Comp. Pract E.*, vol. 26, no. 4, pp. 907-916, Mar, 2014.

[16] N. Huang, B. K. Shoichet, and J. J. Irwin, "Benchmarking sets for molecular docking", *J. Med. Chem.*, vol. 49, no. 23, pp. 6789-6801, Oct, 2006.

[17] 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.

[18] O. Trott and A. J. Olson, "AutoDock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading", *J. Comput. Chem.*, pp. 455-461, Jun, 2009.

[19] 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.

[20] Liferay Inc. "Liferay" [Online]. Available: https:liferay.com. [Accessed: 7 Mar 2017]

[21] M. Kozlovszky, K. Karóczkai, I. Márton, P. Kacsuk, and T. Gottdank, "DCI bridge: executing WS-PGRADE workflows in distributed computing infrastructures", in *Science Gateways for Distributed Computing Infrastructures*, Springer, 2014, pp. 51-67.

[22] S. J. Taylor, T. Kiss, G. Terstyanszky, P. Kacsuk, and N. Fantini, "Cloud computing for simulation in manufacturing and engineering: introducing the CloudSME simulation platform", in Proceedings of the 47th Annual Simulation Symposium, ANSS 14, Tampa, FL, USA, 13-16 Apr 2014, A. Tolk, Ed. SCS, 2014. pp. 12-19.

[23] CloudBroker GmbH. "CloudBroker Platform". [Online]. Available: http://cloudbroker.com/platform/. [Accessed: 7 Mar 2017]

[24] J. J. Irwin and B. K. Shoichet, "ZINC-a free database of commercially available compounds for virtual screening", *J. Chem. Inf. Model.*, vol. 45, no. 1, pp. 177-182, Dec, 2004.

[25] Cloudsigma Holding AG. "Cloud servers & Hosting". [Online]. Available: https://www.cloudsigma.com/. [Accessed 7 Mar 2017]

[26] P. Kacsuk, J. Kovacs, Z. Farkas, A. C. Marosi, G. Gombas, and Z. Balaton, "SZTAKI Desktop Grid (SZDG): a flexible and scalable desktop grid system", *J. Grid Comput.*, vol. 7, no. 4, pp. 439-461, Dec, 2009.