Biochemical Application Porting by Interoperating Personal and IaaS Clouds

Tibor Barat[1] and Attila Kertesz[2,1]

[1] University of Szeged, Department of Software Engineering
H-6720 Szeged, Dugonics ter 13, Hungary
Barat.Tibor@stud.u-szeged.hu
[2] MTA SZTAKI Computer and Automation Research Institute
H-1518 Budapest, P.O. Box 63, Hungary
kertesz.attila@sztaki.mta.hu

Abstract. Researchers of various disciplines ranging from Life Sciences and Astronomy to Computational Chemistry, create and use scientific applications producing large amount of complex data relying heavily on compute-intensive modelling, simulation and analysis. Cloud Computing has reached a maturity state and high level of popularity that various Cloud services have become a part of our lives. The goal of this research is to apply this new technology to support and ease the execution of scientific applications and manage the produced user data in a convenient way, accessible from different places and devices. In this paper a biochemical application is examined as a use case for exemplifying our approach. We provide a general way for porting this biochemical application to an interoperable, heterogeneous environment consisting of Infrastructure and Personal (i.e. storage) Clouds. Our approach stores the application data in Personal Clouds and produce this data in Infrastructure Clouds in an autonomous way. Finally, we evaluate the ported application by using Ubuntu One, Dropbox, OpenNebula and Amazon. Our results show that users can benefit from this approach, and utilize data and infrastructure Cloud services in a transparent and efficient way.

1 Introduction

Researchers of various disciplines ranging from Life Sciences and Astronomy to Computational Chemistry, create and use scientific applications producing large amount of complex data relying heavily on compute-intensive modelling, simulation and analysis. Nowadays Cloud Computing has reached a maturity state and high level of popularity that various Cloud services have become a part of our lives. These services are offered at different Cloud deployment models ranging from the lowest infrastructure level to the highest software or application level. Within Infrastructure as a Service (IaaS) solutions we can differentiate public, private, hybrid and community Clouds according to recent reports of standardization bodies [5]. The previous two types may utilize more than one

L. Lopes et al. (Eds.): Euro-Par 2014 Workshops, Part I, LNCS 8805, pp. 275–286, 2014.

Cloud system, which is also called as a Cloud federation [7]. One of the open issues of such federations is the interoperable management of data among the participating systems. Another popular family of Cloud services is called Cloud storage services or Personal Clouds. With the help of such solutions, user data can be stored in a remote location, in the Cloud, and can be accessed from anywhere. Mobile devices can also benefit from these Cloud services: the enormous data users produce with these devices are continuously posted to online services. The aim of our research is to propose an approach that interoperates Infrastructure and Personal (i.e. storage) Clouds, and to develop a solution that manages application data in Personal Clouds and process it in Virtual Machines (VM) of Infrastructure Clouds in an autonomous way.

In this paper we examine a biochemical application that generates conformers of flexible molecules, here a tetrapeptide (Tyr-Pro-Phe-Phe-NM2), by unconstrained molecular dynamics at high temperature to overcome conformational bias then finishes each conformer by simulated annealing and energy minimization to obtain reliable structures. The main contributions of this paper are: (i) to provide a way for porting biomedical applications into a heterogeneous environment consisting of Infrastructure and Personal Clouds, and executing them in an interoperable and autonomous way, (ii) to propose a solution based an a biochemical use case of a real-world application that manages Ubuntu One, Dropbox, OpenNebula and Amazon, and (iii) to evaluate this application to determine the performance of the proposed solution.

Regarding related works in the area of application porting to Grids and Clouds, Valverde in [9] has already shown, how to execute TINKER binaries [11] in EGEE Grids [10], but this solution used only a low-level, command line interface. On the contrary, we propose a general, high-level solution using a portal framework to provide an easily accessible graphical user interface for non-Grid expert users. In our previous work [6] we have shown how to port a biochemical application to Grids, Supercomputers and Infrastructure Clouds. In this work we take a step forward, and incorporate Personal Clouds to the execution environment to store application data, and to provide a solution more convenient and transparent for users.

The remainder of this paper is as follows: Section 2 presents the considered biochemical application; Section 3 describes our approach for autonomous application execution with interoperating Infrastructure and Personal Clouds. Finally, Section 4 discusses the performed evaluations, and the contributions are summarized in Section 5.

2 The TINKER Conformer Generator Application

The application (shown in Figure 1) generates conformers by unconstrained molecular dynamics at high temperature to overcome conformational bias (T) then finishes each conformer by simulated annealing and/or energy minimization to obtain reliable structures. The parameter files contain reference for the molecular force field (in the present case Amber99), vacuum/implicit water (here

GBSA) environment, target temperatures, etc. The aim is to obtain conformation ensembles to be evaluated by multivariate statistical modeling. It uses the TINKER library [11] for molecular modeling for QSAR studies for drug development. The target end users are biologists or chemists, who need to examine molecule conformers with the TINKER package.

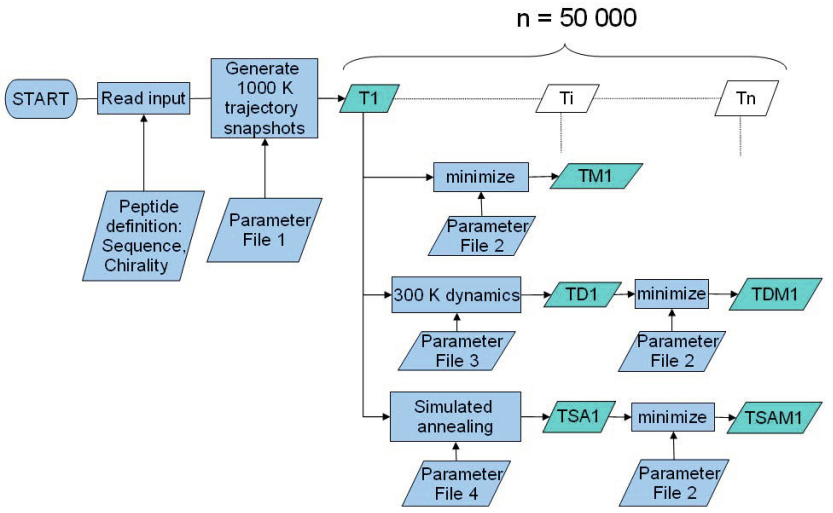

Fig. 1. TINKER Conformer Generator application

The conformer generation algorithm in its present form comprises five different conformer finishing methods:

1. minimizing the initial conformational states generated at high temperature (TM),
2. performing a short low temperature (e.g. 300 K) dynamics with the high temperature conformations to simulate a low temperature thermodynamical ensemble (TD),
3. minimizing the above low temperature states (TDM),
4. cooling the high temperature states by simulated annealing, e.g. to 50 K, or completely to 0 K (TSA),
5. minimizing the annealed states (TSAM).

The reason why to generate the conformational states or conformers (which are conformational states at some local energy minima) is to investigate which of them suits better for the subsequent multivariate statistical modeling (namely quantitative structure-activity relationships studies, QSAR), then the algorithm may be simplified. Our most recent successful QSAR modeling makes use of the TSAM structures which is the most computationally costly method, but may serve as a reference method to obtain the most reliable thermodynamical

ensembles. Regarding execution times, Table 1 shows the average run times of the various methods in the vacuum environment used in the application on a single 2GHz CPU machine (the abbreviations discussed in this paragraph correspond to the ones shown in Figure 1). If we use the implicit water (GBSA) environment, the execution of the different steps takes 1,5 times longer. This means that the execution of the whole application takes around 5-8 days.

Table 1. Execution times of the main steps of the application on a single 2GHz CPU machine

Step	Execution time (hours)
T	13
TM	28
TD	3
TDM	28
TSA	26
TSAM	28

3 An Approach for Autonomous Application Execution over Infrastructure and Personal Clouds

Besides IaaS Cloud solutions the largest amount of user provided data are stored at Cloud storage services also called as Personal Clouds [5,3]. Their popularity is accounted for easy access and sharing through various interfaces and devices, synchronization, version control and backup functionalities. The freemium nature [17] of these services maintain a growing user community, and their high number of users also implies the development of other higher level services that make use of their cloud functionalities. To overcome the limits of freely granted storage, users may sign up to services of different providers, and distribute their data manually among them.

In this work, we have chosen Ubuntu One [13] as a Personal Cloud to design our proof-of-concept implementation. This Cloud service is available primary for Ubuntu users, and makes able to store, synchronize and share user data in the Cloud. It is operated by Canonical [14], and a newly registered user can use 5 GBs free storage. It also provides a web-based graphical interface called Ubuntu One Dashboard (see Figure 2), accessible by a Launchpad account.

In order to access files stored in Ubuntu One from an API, we used REST API [15] and the Oauth standard [21]. In this way we can refrain from sending user names and passwords with each request, instead we can authenticate ourselves by so called tokens, consisting of a single string. The main steps for acquiring and using these tokens is depicted in Figure 3.

By issuing an API call we can get information for a user account on the following data: username, maximum capacity of the usable storage, root path and paths of user created folders. Uploaded files are called as *nodes*. Once a

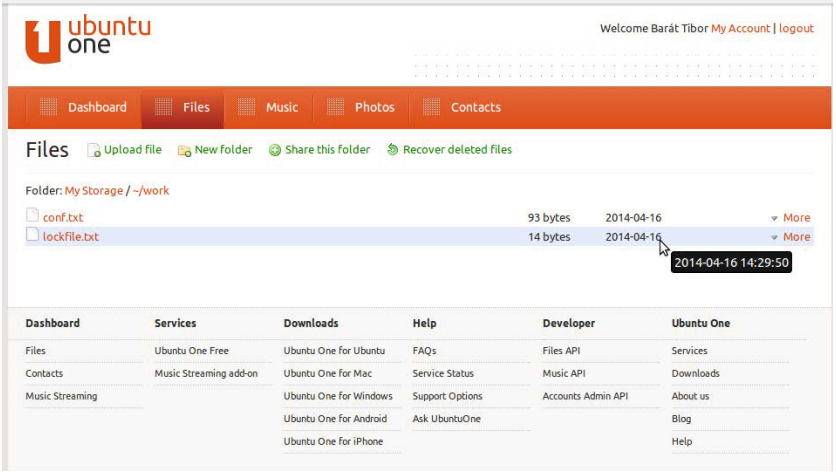

Fig. 2. Ubuntu One Dashboard

node is queried by providing its path, we can get additional information on its availability (private/public), its parent directory, node unique id, creation and latest modification date and a generation number (useful for version control). The *GET* command provides a list of public files, in a JSON format. We used the U1FileAPI library [16] for implementing the communication between the Infrastructure and Personal Clouds. In this way we applied methods as single asynchronous threads.

Our proposed solution shown in Figure 4. By using this approach, users only need to make available his/her data in a Personal Cloud, and to specify with a configuration file the order of data processing (by linking VM methods to data). Once this configuration file is available and at least one VM (executing the necessary service for processing user data) is running in an IaaS Cloud, the autonomous data processing starts and goes on till all data is processed. Steps 1 to 5 (in Figure 4) denote communication between an Infrastructure and a Personal Cloud:

- In Step 1, the configuration file is downloaded to a VM, and the first yet not reserved task is selected.
- In Step 2, the modified configuration file is uploaded back to the storage containing the new reservation.
- In Step 3, the VM downloads the data to be processed in the selected task.
- In Step 4, once the data processing is finished, the results are uploaded to the storage.
- And finally in Step 5, the configuration file is refreshed denoting the successful execution of the selected task.

In the followings we describe our proof-of-concept implementation using the TINKER Conformer Generator (TCG – introduced in Section 2) application

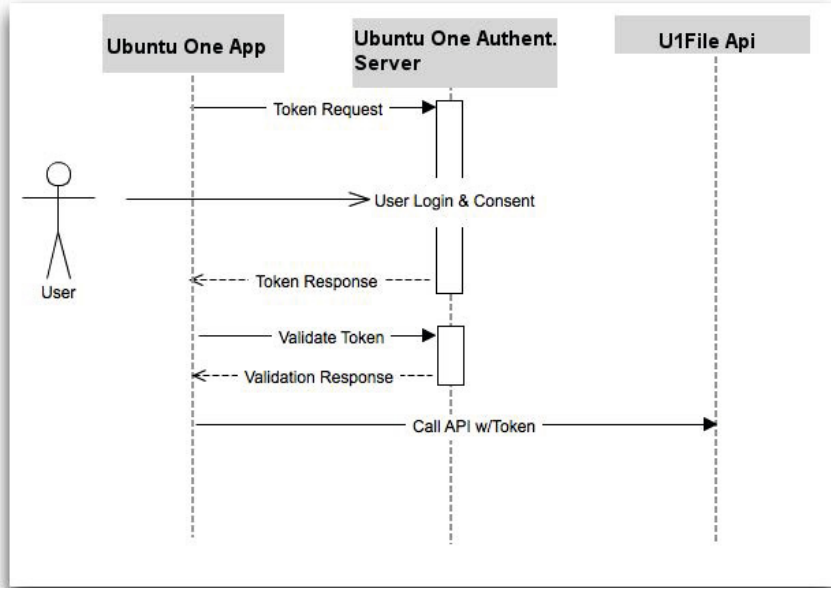

Fig. 3. OAuth usage steps

as a use case. We used three scripts managed by a Java web application to execute TCG in a VM: the master script (*master.sh*) to perform the initial conformer generation (task T in Figure 1), the worker script (*worker.sh*) to perform additional conformer finishing methods for 1000 conformers at a time (including TM, TD, TDM, TSA and TSAM tasks, selection is based on the input data), and the uploading script (*upload.sh*) to compress the sub-results to a single result file. In this way the execution of the ported TCG consists the execution of a master task in the first phase, followed by running 50 worker tasks for processing the 50000 conformers (possibly in parallel) in the second phase, finally calling the uploading script to create the final result in the third phase.

The greatest challenge in porting TCG to this environment was to manage the configuration file properly for the maximum 50 competing parallel workers. A synchronization problem occurs when more than one VM try to modify properties of our configuration file (typically for reserving a task). This problem is exemplified in Table 2 for two competing threads A and B that both try to increment the value of a variable. Thread A reads the value of the variable in Step 2, while Thread B in Step 3. In Steps 4 and 5 they increment the read value in parallel, and write it back in Steps 6 and 7. After two incrementation the value of the variable should be 2 instead of 1.

To solve this issue, first we used the (*listDelta*) method provided by Ubuntu One to track file changes denoted by incremental generation numbers (instead of time stamps) – this is a unique capability of Ubuntu One. In this way we could synchronize VM access to the configuration file by locking it for Steps 1, 2 and

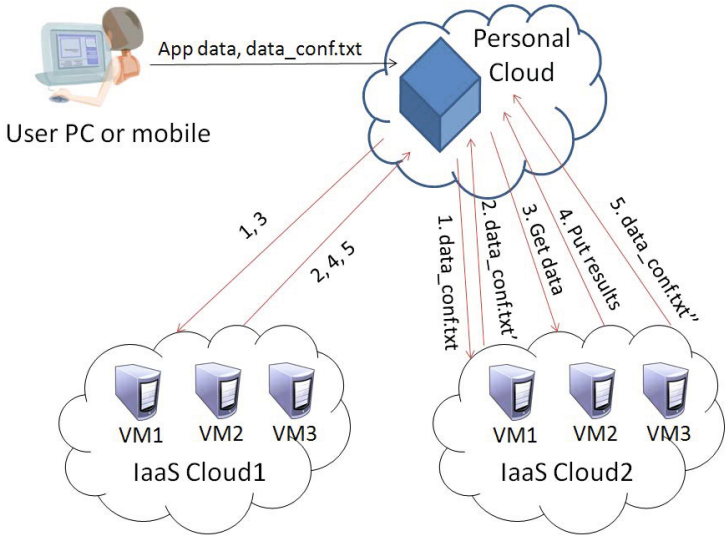

Fig. 4. The proposed solution for interoperating IaaS and Personal Cloud services

Table 2. Synchronization problem

Time	Thread A	Thread B	Value
1	-	-	0
2	read	-	0
3	-	read	0
4	increment	-	0
5	-	increment	0
6	write	-	1
7	-	write	1

5 (to avoid the selection of the same task by multiple VMs). Unfortunately this solution gave a poor performance according to our evaluations (caused by unexpected delays of the *readLine* method of the U1FileAPI library [16]), therefore we have chosen a different solution that can be used by any Personal Clouds, not only by Ubuntu One.

In order to find a better solution for synchronization we turned our attention to Amazon SimpleDB [18]. It provides access to a non-relational database running in the Amazon Cloud. We used its free service providing up to 1 GB data transfer over 25 hours of usage. With this solution we could easily manage concurrent access to our configuration file now moved to this Cloud service. The value of a variable stored in SimpleDB can be incremented in three steps:

1. Get the value of the variable from SimpleDB,
2. increment this value,
3. and finally put the new value instead of the old one in the database.

In order to avoid the above mentioned synchronization problem, we use the so called conditional put or conditional update of SimpleDB in the third step. With this update before writing back the incremented value we can check if the value has been modified since we read it. If this is the case, we won't update the value, otherwise we do write it back. Of course, if the value has been changed by someone else, we need to perform the first two steps again (reading out and incrementing the correct value) till we are able to write our modification back to the database. The next section details our performed evaluations.

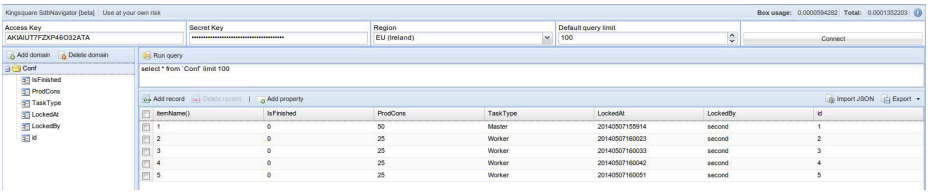

Fig. 5. Configuration parameters in the SdbNavigator tool

4 Evaluation

We have performed our evaluations by using a private IaaS Cloud based on OpenNebula [20]. It has been developed by a national project called SZTAKI Cloud [12], which was initiated in 2012 to perform research in Clouds, and to create an institutional Cloud infrastructure for the Computer and Automation Research Institute of the Hungarian Academy of Sciences. Since 2013 it operates in experimental state, and since 2014 it is in production state available for all researchers associated with the institute. It runs OpenNebula 4.4 with KVM, and controls over 440 CPU cores, 1790 GBs of RAM, 66 TBs shared and 35 TBs local storage for serving an average of 250 Virtual Machines (VM) per day for the last month.

The ported TCG application has been deployed in VMs started at SZTAKI Cloud by a desktop application using the Amazon AWS API [19]. With this tool we can deploy a certain number of VMs in the SZTAKI Cloud that start the TCG application in a web service. First these TCG instances (capable of behaving as masters, workers or uploaders) connect to Ubuntu One and to the Amazon SimpleDB service and query the configuration parameters stored there in a loop till there is any task to perform.

We can also follow the changes of the configuration parameters of the ported TCG application with the SdbNavigator interface tool, which is a free and open source Amazon SimpleDB Administration Interface [22]. A screenshot taken during our evaluation is shown in Figure 5.

To reserve a task, an instance writes its VM identifier and the time of reservation to SimpleDB (to the *LockedBy* and *LockedAt* fields respectively – shown in Figure 5). After reservation, depending on the type of the actual task (*TaskType*)

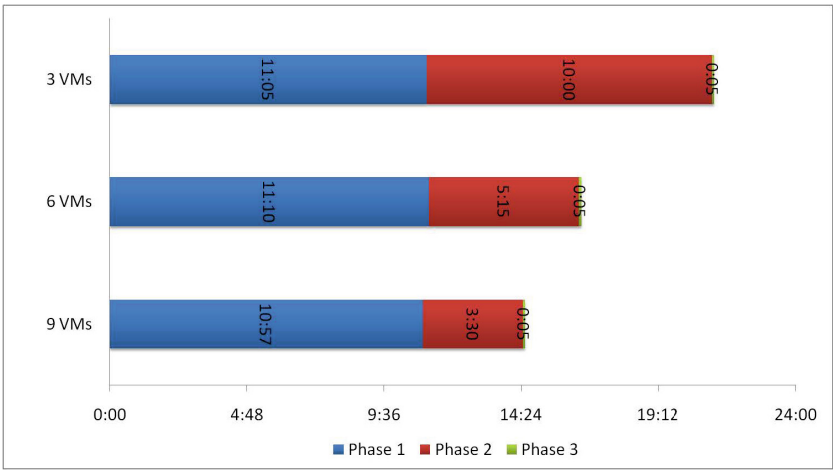

Fig. 6. Evaluation results with Ubuntu One and OpenNebula

the instance downloads the input files from Ubuntu One, and runs the corresponding script.

We have performed the evaluations in three rounds by deploying 3, 6 and 9 VMs at SZTAKI Cloud each having 4 virtual CPUs and 4 GBs of memory. In the first phase, a master task is executed by one VM (the rest of the VMs are waiting for worker tasks), then worker tasks are started in parallel in the second phase, finally an upload task is performed in the third phase to create the final result file. The evaluation results are shown in Figure 6. From these results we can clearly see that by increasing the number of deployed VMs the total execution time is decreasing.

To increase heterogeneity, and to show a scenario when academic and commercial IaaS Clouds are interoperated through a Personal Cloud, we created another evaluation by using Dropbox, OpenNebula and Amazon. We used the same template configuration for OpenNebula (denoted by ONe in the figure) to start 3 VMs in SZTAKI Cloud, and for Amazon (denoted by AM) we also started 3 VMs with Linux Micro instances. It took around 10 minutes to perform the initial input data transfers and to deploy the IaaS VMs to start with Phase 1. We measured nearly the same performance as in the previous round with 6 OpenNebula VMs. The result is shown in Figure 7. Detailed measurement results for Phase 2 of this experiment is depicted in Figure 8. In these diagrams we can see how the VMs of different IaaS systems competed for tasks, and how long it took them to compute these tasks in total (the curve marks the start time of task computations by VMs – except for the first task, which started at 0:00). In both evaluations, the data transfer time between the Personal and the IaaS Clouds were negligible (up to 1 or 2 minutes).

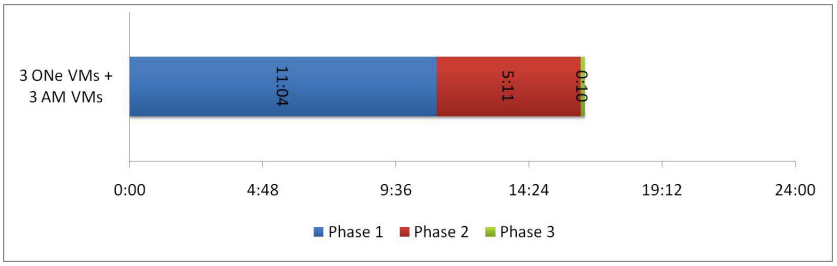

Fig. 7. Evaluation results with Dropbox, OpenNebula and Amazon

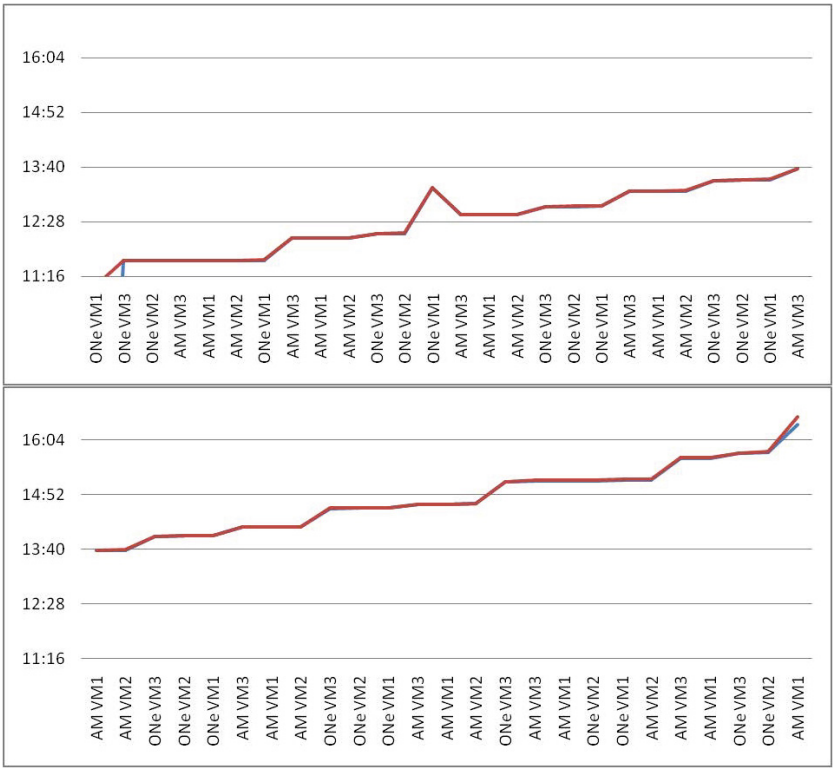

Fig. 8. Detailed evaluation results for Phase 2

5 Conclusion

Cloud Computing has reached a maturity state and high level of popularity that various Cloud services have become a part of our lives. By using mobile devices, users produce enormous data that need to be processed. The aim of our research

was to develop a solution that manages user application data in Personal Clouds and process it in VMs of IaaS Clouds in an autonomous way.

In this paper we exemplified our approach by porting a biochemical application into a heterogeneous environment consisting of infrastructure and Personal Clouds, and executing it in an interoperable and autonomous way. With the evaluation of this real-world application by interoperating Ubuntu One, Dropbox, OpenNebula and Amazon, we have shown that users can benefit from our proposed approach and utilize Personal and IaaS Cloud services in a transparent and efficient way. Our future work aims at investigating the applicability of the proposed solution in a hybrid Cloud-Desktop Grid environment, and generalizing the proposed porting solution and developing mobile clients to further ease the management of application executions.

Acknowledgment. The research leading to these results has received funding from the EU FP7 IDGF-SP project under grant agreement 312297.

References

1. Bozman, J.: Cloud Computing: The Need for Portability and Interoperability. IDC Executive Insights (August 2010)
2. Dillon, T., Wu, C., Chang, E.: Cloud Computing: Issues and Challenges. In: Proc. of the 24th IEEE International Conference on Advanced Information Networking and Applications, pp. 27–33 (2010)
3. Fraunhofer Institute for Secure Information Technology. On THE Security of Cloud Storage Services, SIT Technical reports (March 2012), http://www.sit.fraunhofer.de/content/dam/sit/en/documents/Cloud-Storage-Security_a4.pdf
4. Gagliardi, F., Muscella, S.: Cloud Computing – Data Confidentiality and Interoperability Challenges. In: Book: Cloud Computing, Computer Communications and Networks, pp. 257–270. Springer, London (2010)
5. Jeffery, K., Neidecker-Lutz, B.: The Future of Cloud Computing, Opportunities for European Cloud Computing beyond 2010. Expert Group Report (January 2010)
6. Kertesz, A., Otvos, F., Kacsuk, P.: A Case Study for Biochemical Application Porting in European Grids and Clouds. Special Issue on Distributed, Parallel, and GPU-Accelerated Approaches to Computational Biology, Concurrency and Computation: Practice and Experience, Early View (August 2013)
7. Kertesz, A.: Characterizing Cloud Federation Approaches. In book: Cloud Computing - Challenges. In: Mahmood, Z. (ed.) Limitations and R&D Solutions. Springer Series on Computer Communications and Networks (accepted in 2014)
8. Kertesz, A.: Legal Aspects of Data Protection in Cloud Federations. In: Nepal, S., Pathan, M. (eds.) Book: Security, Privacy and Trust in Cloud Systems, Signals & Communication, pp. 433–455 (2014)
9. Valverde, J.: Simplifying job management on the Grid. EMBnet. News 14(2), 25–32 (2008)
10. Enabling Grids for E-sciencE (EGEE) project (June 2011), http://public.eu-egee.org/
11. Tinker molecular modelling package (May 2011), http://dasher.wustl.edu/tinker/

12. The SZTAKI Cloud project website (May 2014), http://cloud.sztaki.hu/en/home
13. Ubuntu One (May 2014), https://one.ubuntu.com/dashboard/
14. Canonical Ubuntu website (June 2013), http://www.canonical.com/products
15. REST API (May 2014), https://one.ubuntu.com/developer/files/store_files/cloud
16. Ubuntu One Files Java library (U1FileAPI) (May 2014), https://one.ubuntu.com/developer/files/store_files/android
17. Wikipedia – Freemium utilization (April 2014), http://en.wikipedia.org/wiki/Freemium
18. Amazon SimpleDB (March 2014), https://aws.amazon.com/simpledb/
19. Amazon AWS documentation (March 2014), https://aws.amazon.com/documentation/
20. OpenNebula website (March 2014), http://opennebula.org/documentation
21. Oauth standard documentation (March 2014), http://oauth.net/documentation/
22. SdbNavigator interface (May 2014), http://www.kingsquare.nl/sdbnavigator