Cloud computing for fast prediction of chemical activity

Jacek Cała, Hugo Hiden, Paul Watson, Simon Woodman
School of Computing Science
Newcastle University
Newcastle upon Tyne, UK
{jacek.cala | h.g.hiden | paul.watson | s.j.woodman}@ncl.ac.uk

***Abstract*—Quantitative Structure-Activity Relationships (QSAR) is a method to create models that can predict certain properties of compounds. Because of the importance of QSAR in designing new drugs, ability to accelerate this process becomes crucial. One way to achieve that is to be able to quickly explore the QSAR model space in the search for the best models. The cloud computing paradigm very well fits such a scenario, thus we designed and implemented a tool for exploration of the model space using our e-Science Central platform supported by the cloud. We report on scalability achieved and experiences gained when designing this system. The acceleration obtained is much beyond what existing QSAR solutions can offer, which opens potential for new interesting research in this area.**

Quantitative Structure-Activity Relationships, machine learning, cloud computing, scalability, performance evaluation

I. INTRODUCTION

In the search for new anti-cancer therapies, the family of kinase enzymes are important biological targets since many are intimately connected to cell division and other important maintenance functions. The scientists use a method known as Quantitative Structure-Activity Relationships (QSAR) [5] to mine experimental data for patterns that relate the chemical structure of a drug to its kinase activity. If a successful QSAR model can be derived from the experimental data then that model can be used to focus new chemical synthesis, and by creating QSAR models for more than one set of results, for different kinases, the new drugs can be designed to be selective.

Because of the importance of QSAR, the Chemists behind our collaboration had developed the Discovery Bus [2] — an infrastructure to automatically create new and update existing QSAR models as new data or modeling techniques became available. The Discovery Bus could automatically generate hundreds of models for each property type, and select the best and most valid. This enabled creating a library of predictive models used to design better, safer, more environmentally benign drugs, while at the same time reducing the need for animal experimentation.

In our previous work — project Junior — we used the Windows Azure cloud platform to support the Discovery Bus and reduce the time taken to generate QSAR models from years to weeks [11]. In total, 750,000 new QSAR models were generated and made freely available for anyone to use. Before this project it was thought that it would be impossible to generate these models: the chemists estimated that it would take 5 years to process the vast amounts of chemical activity data that had become available.

Although the achieved acceleration in processing was satisfactory, the main problem we encountered was limited scalability of the system. We decided, therefore, to reengineer the approach adopted previously and build a QSAR modeling engine which can be scaled effectively to hundreds of machines.

The main contribution of this paper is to report on scalability achieved and experiences gained with designing a system for fast prediction of chemical activity in the cloud. In particular, we would like to present a scalable workflow enactment system based on e-Science Central which allows us to use up to 200 processors with nearly 90% of ideal effectiveness. Also, we discuss the key steps that made running our system in the cloud effective.

II. THE APPLICATION: QSAR AND THE DISCOVERY BUS

The underlying theory of QSAR is that activity, such as reactivity or biological response, is related to molecular structure of compounds. Thus, molecules with similar structure will have similar activities. For example, as the number of carbons in alkanes increases, so does their boiling point; this rule can be used to predict the boiling points of higher alkanes. QSAR aims to build models that can correlate the structure with activity. Fig. 1 shows the general methodology of creating and using QSAR models [4].

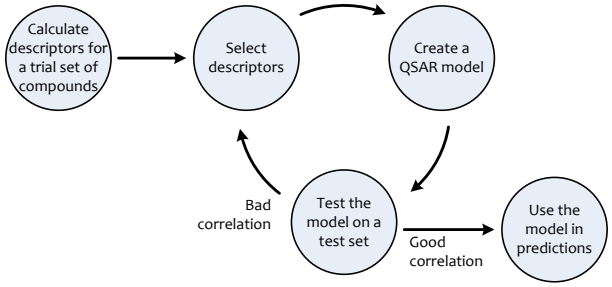

Figure 1. The general process of creating and using QSAR models.

The approach taken by the Discovery Bus to generating predictive models follows the process presented above. However, to optimize it and facilitate its extensibility of the framework some important details has been proposed. Therefore, the actual QSAR process adopted by the Discovery Bus is shown in Fig. 2.

The main path of the process is initiated by a user that provides a number of datasets each of which consists of a set of compounds and their corresponding activity. In the first step each dataset is split into two: a training set that will be used to build models, and a test set used to validate them. Next, for each chemical structure, a set of molecular descriptors is calculated (an example is molecular weight). These are then filtered to remove redundant or irrelevant features so as to speed up learning and increase the generalisability of the results. Next, compounds with related activity and descriptors are used as input by a set of model-building algorithms. The system implemented 4 algorithms for building models, including neural networks, partial least squares, multiple linear regression and classification trees. The best models from each algorithm are selected, based on their performance on the test data, and placed in the database for use by users wishing to predict properties of chemicals.

Importantly, the loop presented in Fig. 1 has been unwound in the actual implementation of the Discovery Bus. The descriptor selection algorithm produces a number of descriptor sets at once. Then each of them is used to initiate a new processing branch with model building as the next step. Similarly, the 'build models' block in the diagram starts a new processing branch for each model builder registered in the system. This branching in data processing can easily be exploited to parallelize the execution.

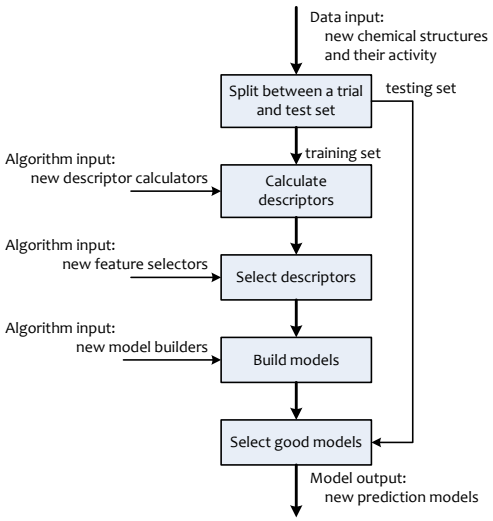

Figure 2. The QSAR process as adopted by the Discovery Bus.

As there are various methods of calculating and selecting descriptors, and building prediction models, the Discovery Bus allowed users to add new algorithms to the system (the 'algorithm inputs' in Fig. 2). Adding an algorithm would reinitiate the QSAR process starting from the intermediate step for which the algorithm was provided and using "historic" data collected in the preceding steps.

III. Cloud Implementation

Whilst the Discovery Bus was the basis for the overall design of the QSAR modeling process, to implement it we used e-Science Central — a cloud-based workflow enactment system. e-SC uniquely combines three recent trends in computing science : (1) Software-as-a-Service — to allow scientists to use the system from a web browser, upload and analyse data and share data and services, all with no need for any installation and deployment by a user; (2) Cloud Computing — to provide resources that scale with the number of users, volume of data and complexity of analysis; (3) Social Networking — to support sharing of data and services and to facilitate user collaboration [12]. With this combination scientists can conveniently design, run and share their analyses on a large scale while being freed from most of the burden of software maintenance.

The architecture of our cloud-based solution is presented in Fig. 3. For evaluation purposes we used Windows Azure to run a complete copy of the publicly available e-SC system (see http://www.esciencecentral.co.uk). The central server executes in two extra large Azure VM instances — one hosts e-SC frontend on top of a JEE application server while the other runs the database engine.

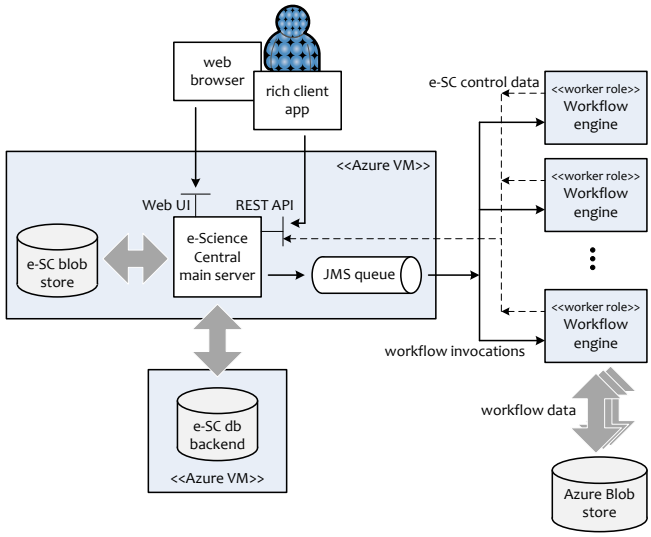

Figure 3. The architecture of e-Science Central in the cloud.

Separately, a number of e-SC workflow engines is running, each in its own Azure worker role node. The engines are connected with the server by a JMS message queue and the REST-based API. Despite Azure offers its own queuing service, to preserve as much consistency with the original e-SC implementation as possible we did not decide to switch the queue services. There is no obvious benefit for such a change. Conversely, as shown by Hill et al. [6], reading rates for the Azure queue service, when accessed by a large number of clients, can drop to as little as 2 messages per second.

A. Execution Model

Users submit their workflows via a web browser or dedicated desktop application. The submission is accepted by the server which creates for it a workflow invocation. The invocation comprises a sequence of service (block) calls. These are either core services available within e-Science Central or custom blocks that the scientist has uploaded.

e-SC supports execution of various service kinds such as Java, R and Octave. They can be as simple as downloading data from blob storage or as complex as building a QSAR model which can consume over one CPU-hour.

System also offers control blocks that can initiate subsequent workflow invocations, and so create invocation chains, trees or even loops. Importantly, workflow invocations are completely independent of each other and may be processed by any of the workflow engines.

1) Dispatch policy: All created workflow invocations are sent to a single message queue from which they are acquired by the engines. Adoption of the work-stealing approach rather than explicit task scheduling, better fits the cloud platforms for at least two reasons. First, worker nodes may be restarted or taken offline anytime during their operation. In Windows Azure this may be caused by service healing or automatic upgrade of the OS. Second, the global invocations queue facilitates adding nodes to and removing them from the resource pool. There is no need for rescheduling tasks when the pool size changes.

2) Workflow execution: A workflow engine acquires one or more workflow invocation messages from the queue. The number of messages retrieved at once depends on the size of the client consumer window and the size of messages. The client-side buffering is a common mechanism implemented by JMS providers and an indirect method to achieve task bundling.

When a workflow invocation is executed, the engine runs the included blocks one by one according to the structure of the flow of data. The definition of a block contains not only the declaration of input ports which the block requires to run but also software dependencies that must be met to start it. For example, a number of blocks in our QSAR scenario need the R runtime environment, and so this requirement is expressed in the block descriptor as a library dependency. Before running a service, any unavailable libraries are downloaded from the server on demand.

Once all software dependencies are met, the engine starts executing a service. To improve security and reliability every block execution involves creation of a dedicated process in the operating system. In the case of Java blocks it is a JVM process, while for R blocks R runtime environment is started.

The overall result of a workflow invocation is sent back to the server as a simple status message (success or failure). Additionally, the server creates for each invocation a dedicated folder where all invocation specific data may be stored; to transfer them e-SC offers a number of I/O blocks.

3) Resource acquisition and release: At this stage work we assumed that engines are running in the cloud before users initiate their workflows. A mechanism for adaptive resource provisioning is left for the future work.

B. Workflow Engines in the Cloud

When moving the system to the cloud, our main concern was on improving its performance while increasing the number of running workflow engines. Three aspects of the engine operation were important in this respect.

Firstly, as the engine is capable of resolving software dependencies automatically, we found that a lot of QSAR blocks' code can be extracted in the form of shared libraries. This helped to minimize the amount of data transferred between engines and the server, and increased capacity of the system, i.e. the number of engines the server could handle.

To further limit communication between the engines and the server we altered the way progress data about workflow execution is reported. By design, the engines send back to the server progress and status information of each service invocation they process; users use this to follow the current state of workflow execution. However, with the growing number of engines the amount of data transferred grew quickly and caused overload of the server. To improve the capacity and performance of the system we added options to minimize the data transferred. Users can decide whether they need additional information from each block. They also can decide to remove the information completely after the successful completion of a workflow invocation. The former enables the number of engines running in parallel to be increased. The latter allows for high performance to be sustained even if a large number of invocations are flowing through the system. Still, if more information is needed, e.g. for debugging purposes, workflows can be configured to retain that for the cost of lower overall performance.

Thirdly, we were able to improve processing speed by running many concurrent workflow invocations in the same engine. The workflow engine processes each invocation using a single execution thread that runs blocks one by one. This better fits a common structure of a dataflow as most of the services depend on just one predecessor or a very small number of preceding blocks. Nonetheless, sometimes the single-threaded execution might introduce underutilization of resources. We observed that in the case of workflows with many I/O-bound blocks.

To overcome this limitation the engine has been extended to accept multiple workflow invocations at once, each running in a separate thread. For our QSAR scenario the best CPU utilization on a single-core worker node was achieved with engines running up to four invocations concurrently.

C. Modeling QSAR Workflows

To model our QSAR scenario we built 12 workflows combined in a graph structure shown in Fig. 4. The workflow design corresponds to the QSAR process presented earlier in Section 2. Once the input data is uploaded into the e-SC data repository, a user can initiate QSAR calculation by invoking the top level workflow. For every input dataset the workflow initiates a single invocation of the prepare descriptors workflow which includes splitting the data between the train and test sets, and calculating and selecting descriptors. Afterwards a number of build and cross-validate workflows is invoked. These workflows consist of model building and validation algorithms taken from the Discovery Bus. As the linear model builder may output one or two models, the cross-validation of linear models has been excluded to a separate workflow. Finally, analysis of cross-validation and model testing are performed. The testing workflows store models together with their metrics in the Azure blob store, so they can be browsed and used for prediction by a separate application.

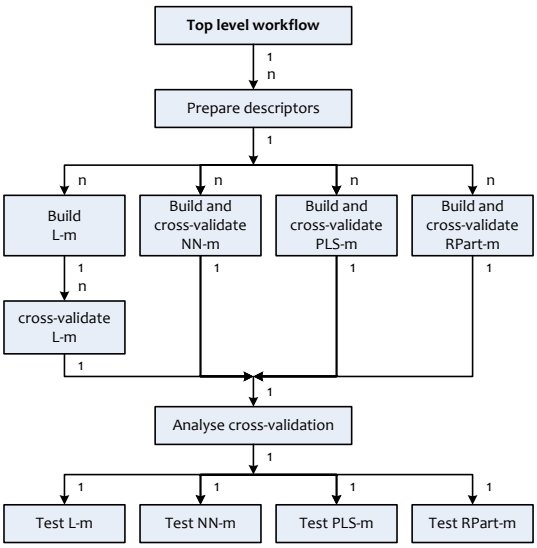

Figure 4. The design of the QSAR scenario workflow.

According to the workflow design, all invocations form a large tree; note that the quantifiers at the arrow ends are 1–1 and 1–n, which means there is no merge step in this scenario. The number of nodes and leaves in the tree varies depending on the actual input data. For example, the prepare descriptors workflow may produce from four to seven feature sets each of which needs to be processed by the following model building workflows. Altogether a single input dataset generates from about 60 to 105 workflow invocations which result in around 1000–1700 block executions.

When designing QSAR workflows, we considered the fact that the basic unit of work in e-SC is a workflow invocation rather than a block execution. Therefore, by combining many short running blocks into a single workflow we were able to reduce communication overheads in runtime. This seems to be one of the important improvements over the Discovery Bus. It is also unlike many other approaches such as Pegasus, Falcon and Hadoop, which operate on a task/operation level. To minimize overheads related with running many short tasks they need to group them together via task bundling [8] or task clustering [1]. Instead, a workflow, being a logical unit designed by user, creates a natural boundary for task collocation so the need for a separate clustering abstraction is reduced.

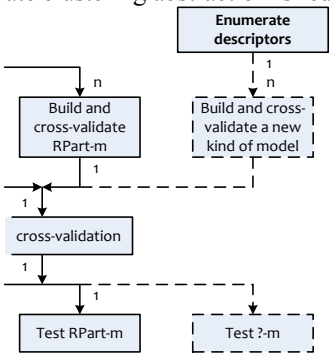

Figure 5. Adding a new model building algorithm can use the enumerate descriptors workflow to reduce the amount of processing needed.

To facilitate adding and experimenting with new models, an additional workflow was added to the system (Fig. 5). It implements one of the algorithm inputs discussed earlier in Section 2. By allowing model building to be started using intermediate data, users can avoid running the time consuming prepare descriptors workflow.

IV. Evaluation

The evaluation of the presented system was run in the Windows Azure platform located in the Western Europe data centre. The server was hosted in two extra large Azure VM instances (2 quad-core AMD Opteron 2.1 GHz, 14 GB RAM each). Workflow engines were deployed in 1–200 small instance worker role nodes (a single core CPU, 1.75 GB RAM each).

Input data for the evaluation purposes were selected from ChEMBLdb (http://www.ebi.ac.uk/chembl) — a database of bioactive drug-like molecules. Initially, the database was curated by using only molecules tested against a well specified biological target and by producing a consistent physical unit of biological activity. Following this, for models with the capacity to be highly predictive, only datasets that contained more than 22 structure-activity values were selected. In result, 11,351 input datasets comprising 1,697,931 small molecules were prepared. Our future goal is to produce models for all these input data but for the purpose of the evaluation just a small subset was used.

Fig. 6 presents the observed speed-up in data processing in relation to the number of workers. As shown, our QSAR scenario scales nearly linearly up to 200 worker nodes. The observed speed-up for 200 workers was 88.2% of the ideal linear speed-up when compared to 20 workers.

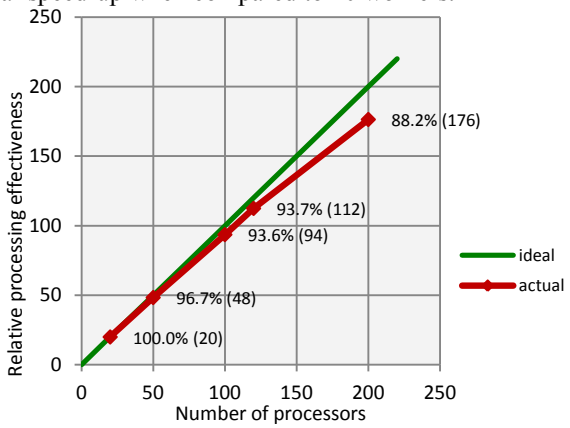

Figure 6. Speed-up in processing QSAR workflows in relation to the number of worker nodes.

We estimate that the achieved processing speed-up will allow us to process the whole 8.5 CPU-months input dataset from ChEMBLdb in less than 35 hours. To the best of our knowledge, it is much beyond what existing QSAR solutions are able to provide.

An important step in achieving the presented scalability was moving almost all data communication from the central e-SC data repository to the Azure blob store. Just this increased the limit on the number of nodes for about 50.

Switching workflow engines to use the Azure storage instead of e-SC was relatively straightforward. We implemented I/O blocks that can transfer data to/from the Azure blob store and changed all e-SC I/O services in the workflows. Importantly, the changes did not involve any part of the e-SC system but were merely limited to running new Azure I/O blocks.

To better understand the performance of our system irrespective of the specific workflow design we conducted a set of additional tests. The tests were done using the sleep N workflow which included one or more "sleep N" blocks that waited a certain number of seconds. We could use that to simulate workflows of various computational complexity.

The first experiment measured the maximum workflow invocation rate. This is an important metric directly related to the maximum system throughput. To estimate the rate we ran up to 20 thousand "sleep 0" workflows. The maximum rate achieved was 55 invocations per second with 200 engines. These results are comparable to systems like Condor, Condor-J2 and Boinc (2, 22 and 93 tasks per second respectively) as reported in [8], but below what Falcon can achieve (from 600 to over 3000 tasks per second with Java and C executors respectively). However, a natural way to improve overall processing performance of our system is to include more than 1 block in a workflow. This is what users almost always do when designing their analyses anyway.

Therefore, despite the experiment with the "sleep 0" workflow revealed a relatively low throughput, running workflows with multiple tasks increases that straightaway. Provided with a sufficiently fast workers and a workflow that consists of ten tasks, the system can potentially reach 550 tasks per second; figure much closer to Falcon with a single dispatcher and the Java-based executor.

An additional side-effect of combining tasks within a single workflow is increase in execution time of workflow invocations which very positively influences effectiveness of processing. Fig. 8 shows the effectiveness for different invocation lengths and different number of worker nodes. The relative processing effectiveness of n workers (RPE_n) was calculated as: $RPE_n = T_1/(nT_n)$, where T_n denotes time needed to process workflow invocations by n workers.

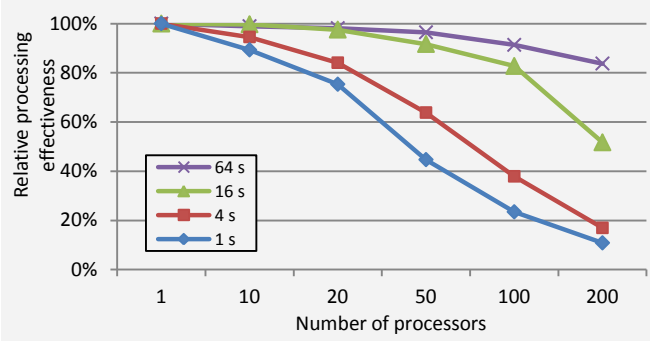

Figure 7. Relative processing effectiveness as a function of the number of worker nodes and invocation length.

As presented in Fig. 8, e-SC with 200 worker nodes exhibits good effectiveness for invocations at least 32 seconds long. For running times below 8 seconds the effectiveness drops quickly at around 50 workers showing that for short invocations the system performs best with no more than 20 processors.

V. Related Work

QSAR modeling is a well established research field with over 40 years of history [10]. A number of approaches and methods to support the modeling exist, yet it is difficult to find systems that consider processing QSAR on larger scale.

QSAR Workbench [7] is a commercially available web-based system developed by Accelrys. It can automate and accelerate QSAR model building by using cluster resources. The system offers a rich set of tools for data preprocessing, analysis, descriptor calculating and model building. Similarly to our use case, the tools can be assembled to build QSAR modeling workflows using a graphical interface.

However, as there is not much information revealed regarding the performance of QSAR Workbench, we could only found that it is able to reduce modeling time from days to hours. In contrast, we report nearly 180 times processing speed-up, i.e. months to hours or years to days reduction.

AutoQSAR [13] is a proprietary system developed at AstraZeneca in collaboration with Accelrys. Its main purpose is to automatically create, evaluate and maintain QSAR models. The main idea of the system is similar to the Discovery Bus — to improve prediction accuracy of models by updating them with newly acquired data. To calculate models AutoQSAR employs the Sun Grid Engine platform [14]. Unfortunately, very limited information about the design and performance of the system does not allow us to compare AutoQSAR with our solution.

Apart from systems specialized in QSAR modeling, there exists a lot of workflow management systems, and high-throughput, high-performance and many-task computing platforms which could potentially be used to implement the QSAR modeling pipeline (for an overview see e.g. [3] and [9]). The most prominent example of these is Falcon that implements the many-task computing (MTC) approach [8]. It has been used on systems in the range from clusters to supercomputers with up to 160 thousand processors, which proves its excellent processing and scaling capabilities.

Although our system does not scale to the extent Falcon can do, we believe that it may be interesting for several reasons. The maximum service invocation rate we achieved was 300 blocks per second; a figure better than many other existing solutions. The system offers good effectiveness when running hundreds of nodes. It can run with over 80% of ideal effectiveness with 100 and 200 workers when workflow invocations are longer than 16 and 64 seconds respectively. And the effectiveness was confirmed running our QSAR modeling scenario — 88.2% for 200 nodes.

Definitely, a valuable feature of the system is that as the basic unit of work it uses a workflow rather than task invocation. Not only does it increase the run time of an invocation, which improves effectiveness, but also it allows for fast data transfer between the subsequent services. Unlike Falcon and other solutions based on task scheduling, blocks in our system communicate using local disk rather than shared file system; an important property for cloud-based systems in which users also pay for network transactions.

VI. Conclusions and Future Work

We presented a fast and scalable way to perform the exploration of the QSAR model space. The acceleration achieved is much beyond what existing solutions can offer. Overall, the cloud computing model is a very good fit for the presented scenario. After processing of the 11 thousand input datasets from ChEMBLdb, further efforts with QSAR modeling will require much less resource. The database is regularly updated, thus we can extract several hundred new input datasets every three months. This is less than 10% of the current database size, and so we will need a fraction of the resources to process it effectively. Also, the development of new model building and descriptor selection algorithms can be tested on a relatively small part of the input sets and for only the most promising ones the whole input data will be applied. Importantly, introducing new model building algorithms can reuse data from previous invocations reducing the need for large computing resources even more.

Using e-Science Central, we were able to migrate the existing QSAR modeling pipeline and run it effectively in the cloud. Meanwhile several important lessons were learnt.

Reducing the amount of data transfers between the server and the engines was of major impact on scalability and processing effectiveness. We used the Azure blob store that proved to be scalable enough to overcome a bottleneck related to communication with the central e-SC data repository. Switching to the Azure storage was as simple as adding to the palette of existing e-SC blocks a few new I/O services (100–150 lines of Java code each) and changing the existing I/O blocks in all related workflows.

Moreover, by expressing service software dependencies we could extract most of QSAR blocks' code in the form of shared libraries. This minimized overheads related to downloading service code by the engines. Further reductions in the amount of data transferred were possible by enabling users to turn off sending blocks' status data after completion of a workflow invocation. Users can decide whether they need faster execution or more detailed status information.

Finally, e-Science Central uses workflow invocation as the basic unit of work. Workflows are usually designed to be a consistent and logical part of the whole scientific analysis. Therefore, they create a natural boundary for service collocation which allows improving processing throughput.

The current design of the system reaches scalability limitation at about 200 worker nodes. Running more workers causes overload to the data store VM, results in execution failures and lowers overall system performance. Whilst, for the QSAR use case 200 nodes gave more than satisfactory results, in the future we would like to remove this limitation.

The presented work may be further extended to address a number of interesting research venues. We plan to improve the process of QSAR modeling by employing more model building algorithms. Also, despite that for our QSAR use case the processing performance is more than sufficient, we want to further improve the scalability and effectiveness of the system. Finally, we want to investigate methods for adaptive allocation of cloud resources in order to decrease costs related to system operation while giving users ability to run their experiments with maximum speed if needed.

Acknowledgment

This work was supported by EU FP7 project VENUS-C. We would particularly like to thank Juan Vargas and Dennis Gannon and Stian Thorgersen for their support to the project.

References

[1] S. Callaghan et al., "Scaling up workflow-based applications," Journal of Computer and System Sciences, vol. 76, no. 6, pp. 428–446, Sep. 2010.

[2] J. Cartmell, S. Enoch, D. Krstajic, and D. E. Leahy, "Automated QSPR through Competitive Workflow.," Journal of computer-aided molecular design, vol. 19, no. 11, pp. 821–833, Nov. 2005.

[3] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-Science: An overview of workflow system features and capabilities," Future Generation Computer Systems, vol. 25, no. 5, pp. 528–540, May 2008.

[4] E. X. Esposito, A. J. Hopfinger, and J. D. Madura, "Methods for applying the quantitative structure-activity relationship paradigm.," Methods in molecular biology (Clifton, N.J.), vol. 275, pp. 131–214, Jan. 2004.

[5] C. Hansch, A. Leo, and D. H. Hoekman, Exploring QSAR: Fundamentals and applications in chemistry and biology. American Chemical Society, 1995.

[6] Z. Hill, J. Li, M. Mao, A. Ruiz-Alvarez, and M. Humphrey, "Early Observations on the Performance of Windows Azure," in High Performance Distributed Computing, 2010, pp. 367–376.

[7] C. Luscombe, "QSAR Workbench: Guided QSAR Model Building for nonExperts." The UKQSAR and ChemoInformatics Group, Cambridge, UK, 2011, presentation, available on-line: http://www.ukqsar.org/slides/Nov2011_Luscombe.pdf (accessed 2/Mar/2012).

[8] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster, and M. Wilde, "Falkon: a Fast and Light-weight tasK executiON framework," in Proceedings of the 2007 ACM/IEEE conference on Supercomputing — SC'07, 2007, pp. 1–12.

[9] I. Raicu et al., "Middleware support for many-task computing," Cluster Computing, vol. 13, no. 3, pp. 291–314, Apr. 2010.

[10] A. Tropsha, "QSAR Modeling and QSAR Based Virtual Screening, Complexity and Challenges of Modern," in Encyclopedia of Complexity and Systems Science, R. A. Meyers, Ed. 2009, pp. 7071–7088.

[11] P. Watson et al., "Accelerating Chemical Property Prediction with Cloud Computing." Microsoft Research eScience Workshop, Berkeley, CA, US, 2010, presentation, available on-line: http://www.esciencecentral.co.uk/docs/2010-10.MSFTe-Science-slides.pdf (accessed 2/Mar/2012).

[12] P. Watson, H. Hiden, and S. Woodman, "e-Science Central for CARMEN: science as a service," Concurrency and Computation: Practice and Experience, vol. 22, no. 17, pp. 2369–2380, Dec. 2010.

[13] D. J. Wood, D. Buttar, J. G. Cumming, A. M. Davis, U. Norinder, and S. L. Rodgers, "Automated QSAR with a Hierarchy of Global and Local Models," Molecular Informatics, pp. 960–972, Nov. 2011.

[14] D. Wood, A. Davis, and S. Rodgers, "AutoQSAR Automation of the QSAR Modelling Process," Presentation at UK-QSAR – Autumn Meeting 2011. The UKQSAR and ChemoInformatics Group, Cambridge, UK, 2011, presentation, available on-line: http://www.ukqsar.org/slides/Nov2011_Wood.pdf (accessed 2/Mar/2012).