ScaBIA: Scalable Brain Image Analysis in the Cloud

Ali Gholami[1], Gert Svensson[1], Erwin Laure[1], Matthias Eickhoff[2] and Götz Brasche[2]
[1]*PDC and HPCViz, Royal Institute of Technology,Teknikringen 14, 10044, Stockholm, Sweden*
[2]*Microsoft Research – Advanced Technology Labs (ATL) Europe, Ritterstrasse 23, 52072, Aachen, Germany*

Keywords: Cloud Computing, SPM, Microsoft Azure, e-Science as a Service, Brain Imaging, FMRI.

Abstract: The use of cloud computing as a new paradigm has become a reality. Cloud computing leverages the use of on-demand CPU power and storage resources while eliminating the cost of commodity hardware ownership. Cloud computing is now gaining popularity among many different organizations and commercial sectors. In this paper, we present the scalable brain image analysis (ScaBIA) architecture, a new model to run statistical parametric analysis (SPM) jobs using cloud computing. SPM is one of the most popular toolkits in neuroscience for running compute-intensive brain image analysis tasks. However, issues such as sharing raw data and results, as well as scalability and performance are major bottlenecks in the "single PC"-execution model. In this work, we describe a prototype using the generic worker (GW), an e-Science as a service middleware, on top of Microsoft Azure to run and manage the SPM tasks. The functional prototype shows that ScaBIA provides a scalable framework for multi-job submission and enables users to share data securely using storage access keys across different organizations.

1 INTRODUCTION

Over the last twenty years, cloud computing has emerged as a new business model leveraging the high level abstraction of hardware and software resources but with low capital costs and on-demand scalability of resources to consumers. The economic factors combined with the key characteristics such as *simplicity*, *elasticity* and *availability* make cloud services attractive to many users within a wide variety of organizations. Nowadays, the market is offering a large number of cloud-based services on a global scale for organizations and companies around the world to improve their efficiency and to reduce costs (Hwang et al., 2011). Furthermore, many enterprises and organizations are considering using cloud storage to store their data as a result of the data flood coming from their users or customers.

In this paper we apply cloud computing to the functional analysis of 3D brain imaging data acquired mainly from Magnetic Resonance (MR) scanners – so-called fMRI analysis – and describe the design of a prototype system for that task. The problem at hand is to analyse which part of the brain is activated when subjects perform certain tasks. To find the activation pattern, a number of brain images are taken in a time series for several subjects, both when subjects are performing the task and when they are not doing so. The prototype has been developed based on an open source toolkit called Statistical Parametric Mapping (SPM) which utilizes MATLAB (SPM, 2011). We implemented our prototype using the generic worker (GW) on top of Microsoft Azure to bridge the gap between Platform as a Service (PaaS) and Software as a Service (SaaS) layers (VENUS-C FP7 Project, 2010), since Microsoft Windows is used as a de facto platform by many brain imaging research groups and communities.

2 BACKGROUND

In this section we give an overview of the neuroscience workflow involving SPM and describe the VENUS-C cloud architecture with a focus on the GW component used in our implementation.

2.1 SPM Overview

The goal of the functional analysis of brain images is to find the parts of the brain that are activated when people (subjects) perform certain tasks. Since the signals that can be measured from the brain are noisy and there is considerable variation between

Gholami A., Svensson G., Laure E., Eickhoff M. and Brasche G..
ScaBIA: Scalable Brain Image Analysis in the Cloud .
DOI: 10.5220/0004358003290336
In *Proceedings of the 3rd International Conference on Cloud Computing and Services Science* (CLOSER-2013), pages 329-336
ISBN: 978-989-8565-52-5

individuals, many subjects, and many images of each subject's brain, are required to get any statistically significant results from the analysis. There are also several parameters that need to be varied during the analysis. This means that the analysis normally has to be executed many times with different hypotheses and parameters.

A typical study consists of hundreds to thousands of 3D images of each subject; the resolution of a single image is normally in the order of few millimetres which gives an image size of several MBytes. There are normally 10 to 100 subjects in each study, and it usually takes over a day for a single analysis on a normal PC. Hence there is a need for scalable compute and storage resources.

A typical analysis of brain images generally consists of several steps as shown in Figure 1:

- Re-align (compensate for subject head movement),
- Co-register (align structural and functional images),
- Normalize (transform to standard brain space),
- Segment (remove the scull bone etc. and leave only the brain),
- Filter (remove noise by low-pass filter), and
- Apply statistic methods, which normally use the General Linear Model (GLM).

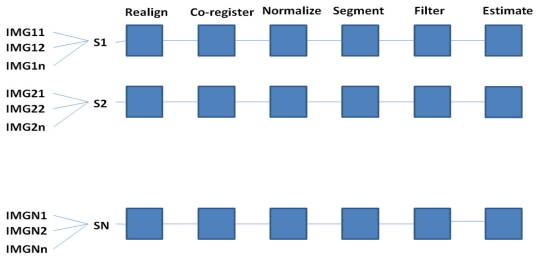

Figure 1: A series of stages to do an fMRI data analysis over N subjects (S_1, S_2, ..., S_N) each subject i containing n images ($IMGi,1$, $IMGi,2$, ..., $IMGi,n$).

After running all stages required for an analysis described in Figure 1, users may try to make inference using different parameters in their model or do Bayesian analysis or several other methods on the results. As an example, Figure 2 illustrates results of the estimation stage – the last process in Figure 1 – to make an inference by a user in a parametric approach.

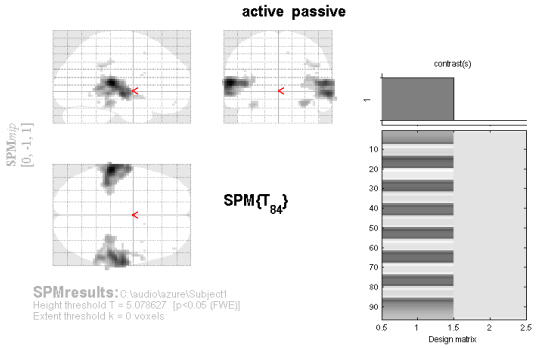

Figure 2: Resulting activation map of an experiment.

In all of these steps, the analyses of different subjects are independent (except for the GLM-step in certain cases). In some steps, like filtering, the analysis of each image is also independent. This implies that it is possible to handle the subjects (and in some cases also the corresponding images) in parallel to speed up the analysis.

The toolkit that is used most for this kind of analysis is SPM, which is an open source toolkit based on MATLAB. The toolkit provides a graphical user interface that allows the user to define and execute MATLAB scripts to run all single steps of the analysis. More advanced users can work directly in MATLAB and define their own scripts. One goal of the prototypic cloud-based implementation described in this paper was to maintain this flexibility.

By using cloud technology, we can speed up individual analyses without having to invest in computing hardware. Running each subject in parallel speeds up the process by a factor that is roughly the number of subjects in the study. Moreover, by using the cloud, we can execute several analyses at the same time with different parameters. The above reasoning suggests that it is more economical to purchase computing capacity on demand, rather than building up a permanent infrastructure that could cope with the peak demands.

Most neuroscience laboratories in the world have made little efforts to share data between researchers in the same laboratory, let alone to share data between different laboratories. Currently, many researchers in the field store their fMRI data and results on the local hard disk or network file system (NFS) home directory that slows down the process of sharing data among users corresponding to different organizations. In addition local storage restricts expanding the storage resources when data volume increases. As a result of this, experiments

may be repeated unnecessarily. Although data sharing was not the focus of this study, the prototype we developed has shown that cloud technology is a convenient way to simplify sharing of data in the neuroscience field. Hence, users will not be restricted to their organization's boundary or technological bottlenecks to access the fMRI storage on-demand.

2.2 VENUS-C

Virtual Multidisciplinary EnviroNments USing Cloud Infrastructures (VENUS-C) was a project from July 2010 to June 2012 in the European Commission's 7th Framework Programme (VENUS-C FP7 project, 2010). The project aimed to develop, test and deploy an industry-quality, highly-scalable and flexible cloud infrastructure for e-Science. The overall goal was to empower the many researchers who do not have access to supercomputers or big grids, by making it easy to use cloud infrastructures. For this to be feasible, the project had to minimize the efforts that such researchers need to spend for development and deployment in order to do computations in the cloud, thereby also reducing the costs for operating the cloud.

2.2.1 VENUS-C Architecture

In order to achieve the goals, the project collected requirements from different scientific use cases and as a result designed a platform that is capable of supporting multiple programming models, such as batch processing, workflow execution or even Map/Reduce (Dean and Ghemawat, 2004) at the same time.

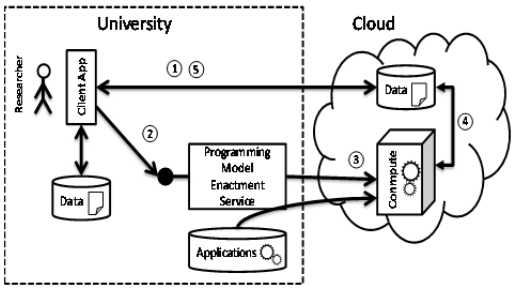

Figure 3: VENUS-C architecture.

Figure 3 illustrates the generalized VENUS-C architecture (VENUS-C Deliverable D6.1, 2011) and shows the basic steps that a researcher must perform in order to use VENUS-C. These steps are independent of the programming model that is used. Firstly the researcher uploads the locally available data to the cloud storage. The next step is to submit a job. So called dedicated Programming Model Enactment Services (PMES) are provided for this purpose. These services enable the researchers to perform tasks such as managing jobs or scaling the resources used in the cloud, while simultaneously shielding the researchers from the underlying cloud infrastructure and the specific implementations of different infrastructures through open grid service architecture – basic execution services (OGSA-BES) compliant interfaces (Foster et al., 2007). OGSA-BES is an open standard for basic execution services and widely used in grid communities for submitting jobs. The third step involves carrying out the required computations. For this, the necessary application and job specific data is transferred to the compute node. After the computation has finished, the fourth step consists of transferring the resulting data to the cloud storage. In the fifth and final step, the researcher can download the results from the cloud to local facilities.

2.2.2 Generic Worker

The GW module (Generic Worker Complete Documentation, 2012) has been developed in the VENUS-C project. Following the general VENUS-C architecture, the GW represents a reference implementation for a batch processing programming model and is available for public download.

The GW is basically a worker process (similar to Windows Service or UNIX daemon processes) that can be started on virtual machines (VMs) in the cloud. Being able to run many VMs at the same time with a GW worker process provides great horizontal scaling capabilities and allows work items to be distributed across the machines according to the user's requirements.

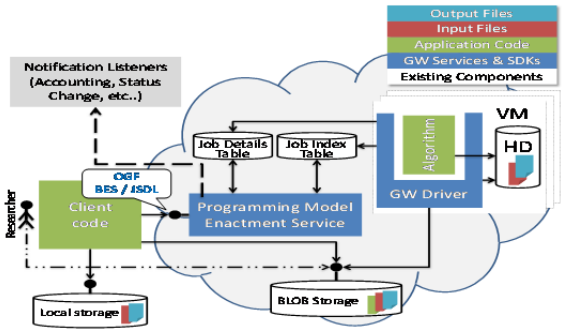

Figure 4: Simplified internal GW architecture.

Figure 4 shows how the GW is designed internally. In a similar way to that described in the

general VENUS-C architecture, the researcher uploads applications and data to storage, which is connected to the Internet so that the GW can also access it. The GW design supports a broad selection of different protocols and storage services. In addition to the data and the application that should be run, the GW also needs a description of this application containing metadata about it. This information allows the GW to understand parameters like input and output files enabling a proper execution of the application by the GW.

Jobs are submitted using the PMES. To make this safe, different security mechanism – such as a Security Token Service (STS) to validate, issue and exchange security tokens based on the well-known WS-Trust protocol (Djordjevic and Dimitrakos, 2005) and username / password – can be used. The PMES stores all the incoming jobs in an internal job queue based on a table (Job Index); an additional table is used for the job details. The GW driver processes continuously look for new jobs in this queue. As soon as a driver process finds a job in the queue, it will pull the job from the queue, and check the application and data storage to find out if everything that is needed is available, namely all the required input data and the relevant application binaries. If these are in place, the job can be executed. The driver process that found the job marks the job as being processed by that particular driver in the "job details"-table (JDT), and starts downloading the input data to the local hard disk of the VM. If application or data are not yet available, the job will be put back into the queue to wait for the missing files. The driver process also checks whether the application is already present on the VM and, if necessary, the application will be downloaded as well. Thus the GW process follows a data-driven pull model, allowing simple workflows where jobs rely on the output of other jobs.

Once the application is available, the driver process retrieves information on how to call the application and then launches it. After the application terminates, the results are made persistent by uploading them to the data storage. Finally, the driver process uses the JDT to mark the job as either completed or failed, depending on the exit code of the application. Researchers who used the PMES client-side notification will be notified about this event. There are several notification-plug-ins available e.g. sending mails or putting messages in a queue for every event. Researchers can also query the PMES to check the current state of a job.

2.3 Related Work

Several research groups made attempts to optimize the execution time of SPM scripts, including parallelizing it. For instance, Parallel SPM (PSMP) is a package that has been developed using Message Passing Interface (MPI) to provide parallelism in SPM (PSPM and MPI, 2011). Beno is another package that can be deployed on a local cluster to run SPM single subject analyses in parallel. (Beno, 2011). However, it creates bottlenecks when parallel nodes try to access the external storage. Both PSPM and Beno rely on multi-CPU clusters to run a single subject in parallel and do not address scalability related to the number of subjects. While these efforts provide increased efficiency for studies on local infrastructures, they lack the resource scalability provided by cloud infrastructures. By using cloud infrastructures analysis can be scaled to computing resources typically not available in local environments. To the best of our knowledge, this work is the first attempt to perform SPM analysis on cloud infrastructures.

3 CLOUD BRAIN IMAGING

As we wanted to make it easy for researchers to use the cloud, we aimed to design a system that would not require the end user to be aware of the complexity of cloud computing or of any dependencies on GW libraries. It was important to preserve the job execution style of SPM, so that users could run their brain imaging jobs using the MATLAB command line. Thus, in our architecture, we opted for a user-friendly interface, with minimal necessary dependencies on third party libraries (to provide a secure communication channel between the GW endpoint and the clients).

3.1 ScaBIA Architecture

To enable users to communicate with and submit brain imaging tasks to the GW, we integrated three main components into our prototype: an application manager, a job manager and a data manager, as illustrated in Figure 5. Client interaction is based on STS to ensure that only authorized users are allowed to register applications and submit jobs. We divided the certificates into two different categories: one for users, and the other for management purposes. In principle, user certificates are used when invoking commands to submit jobs to the GW end-points or

when establishing secure channels and encrypted communication to the Azure cloud platform, e.g. using management certificates for scaling the number of VMs running at the same time (Generic Worker Complete Documentation, 2012).

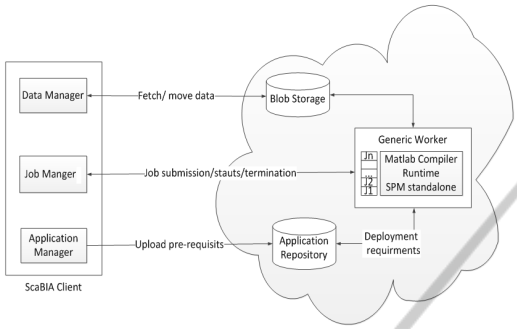

Figure 5: Architectural view of the ScaBIA system.

3.2 ScaBIA Components

Generally, the user requirements for execution of cloud-based SPM can be divided into three major areas. Users need to be able to prepare the execution infrastructure without experiencing difficulties caused by the cloud middleware. Secondly, the users need to be able to execute their jobs and keep track of all submissions. Finally users need to be able to transfer and share data between the cloud storage and their local hard disk securely over the Internet without encountering cross-organizational restrictions introduced by the network services.

3.2.1 Application Manager

Prior to accepting any requests from users, the cloud environment has to set up any mandatory libraries and software. To successfully install an application in GW, the Application Manager (AM) packs all the files that are necessary for running that application into a compressed file. The AM then creates an application description which defines the command line execution format for the application. Next, the AM uploads the application to the application repository that is based on the Azure blob storage. To do this, the AM has to serialize the application description into an XML structure that is uploaded to the application repository. Users need to remember the location of both the application and of the description for job submission purposes. The AM archives different user pre-requisites (for running a library-dependent job) into a single zip-file and uploads it to the pre-defined cloud storage.

3.2.2 Job Manager

To submit SPM jobs, the user provides a job description defining the arguments for the job and their values, in addition to the SPM script created by the user through the SPM GUI (which is the real job that will is be executed). The Job Manager (JM) compresses all the brain images, together with the SPM scripts and job description, to submit to the GW endpoint. The GW job description API implements the job submission description language (JSDL) that is a specification to define submission aspects of jobs such as job name, resource requirements and so on (Savva, 2005). There are some scenarios where data, such as brain images, are already stored in the data storage in the cloud and there is no need to re-transfer the input data. For instance, in a chain of SPM tasks, the results from task N-1 will be used as input for a new task N. In such scenarios, the user only needs to use the JM as a job submitter and provide the location of the input files within the data storage.

Furthermore, the JM provides functions to check the status of submitted jobs and to terminate jobs in any stage of execution. To check the status of jobs, the JM periodically polls the GW with the job id returned from the earlier job submission steps. Thus the JM can notify the job owner when a job is completed. Moreover, the JM is able to get the status of a list of jobs, or of all the jobs in the job table that belong to a specific user.

3.2.3 Data Manager

Each SPM job requires hundreds of granular images that must be present for the job execution. Therefore, we require an effective data transfer solution and also have to be able to download results with names that humans could read. The Data Manager (DM) enables users to upload or download data results from/to their local hard disk, and lets them rename data in the data storage. An excellent example can be a scenario where user completes the analysis stages (Figure 1) and wishes to download the results locally to visualize the results within the SPM toolkit.

Furthermore, in a distributed environment, which is typical for the research world since users are scattered between different organizations, the DM provides a useful facility for sharing brain images between users and cloud applications. The DM acts as a cross-boundary client enabling users to perform create, read, update and delete (CRUD) operations. External users who wish to access the brain imaging

data can easily invoke the client independently of their geographical locations.

Prior to any attempts by users to access the storage, the data owner must grant access privileges to the storage according to users storage access keys through the Azure management portal. Azure storage requires two sorts of access keys known as primary and secondary storage keys to authenticate users. The primary storage key is a mandatory 512-bits symmetric key that will be used as the main authentication key. The secondary storage key is optional to have and it can be generated independently from the primary key to act as backup with similar access rights. A designated host with a management certificate is allowed to add or remove users belonging to different domains.

4 IMPLEMENTATION

The GW provides a Microsoft .NET API for applications to implement all functionality that is needed to execute SPM job successfully. We integrated the GW and SDK assemblies (DLLs) into the MATLAB *R2011a* environment. To relax the licensing issues, we used the MATLAB Compiler Runtime (MCR) Windows 64-bits version *7.15* associated with MATLAB *R2011a* (MCR, 2011).

4.1 Deploying the Generic Worker

As a preliminary step, we deployed SPM using GW on top of Azure to facilitate deployment, initialization, and invoking SPM stand-alone over Azure without any modification of the SPM API. The Windows Azure management portal provides user interfaces to upload security credentials, along with the GW and configurations. The GW is provided in different deployable packages: extra small, small, medium, large and extra-large. We deployed a production-hosted service composed of 20 medium-sized dual core machines and 3.5 Gigabyte memory with a bandwidth of 200 Mbit/s (Microsoft Windows Azure, 2012).

This hosted service also required an XML service configuration file that defines how the hosted service should run, for example, specifying the number of running instances or user and management certificate thumbprints, or other information (such as the job submission to GW end-points). We issued two self-signed OpenSSL certificates, one for management and one for job submission: both based on their distinguished names. We added the thumbprint of these certificates to the hosted service configuration file and uploaded the certificates to the management portal. The uploaded certificate contains both public and private keys in a single file, with a protected private key defined by the user's secret.

4.2 Building the Application

GW instances search for the MCR and standalone SPM dependencies during the initialization process. Therefore, prior to running the GW instances, we have to upload the MCR *v7.15* and standalone SPM for Microsoft Windows 64-bits platform (VENUS-C Software and Document Resources, 2012).

We implemented a MATLAB function to act on behalf of the user to compress the SPM dependencies and to upload them to the application repository using the following command line:

```
install_application(mcr_path,spm_path,
                    app_rep)
```

The first argument of this command is the location of the MCR. The second and third arguments are the SPM standalone location, and the name of the application repository where the user pre-requisites should be stored. After successful installation of the application pre-requisites, the GW will load and install the pre-requisites for all new VMs.

4.3 Job Submission

For job submission purposes, we developed two MATLAB functions. The user enters these on the command line to submit a job, either for a single subject, or for several subjects to run in parallel to ensure scalability for scenarios with a large number of subjects. The signature of the function to submit SPM jobs is as below:

```
submit_job(job_script,data_path,
       output_name, flag)
```

This function needs the following arguments: the real SPM script as the execution job, the path to the brain images, the name to be used for the output results, and a flag that is set in cases of multiple job submission (where an SPM job will be iterated over several subjects within different running GWs). To clarify, the SPM job is a script where the user creates an iteration of a set of instructions over a number of brain images for a group of one or more subjects. The second argument specifies the directory path where the images from that subject reside. Those images need to be uploaded with the submission of the SPM job. This command adapts the local file system names according to the GW

standard by eliminating the root directory and replacing that with the current working path in GW. Figure 6 illustrates the process of creating SPM scripts by the user through the GUI and making them compatible to run on Azure by JM, in respect to the GW file addressing conventions.

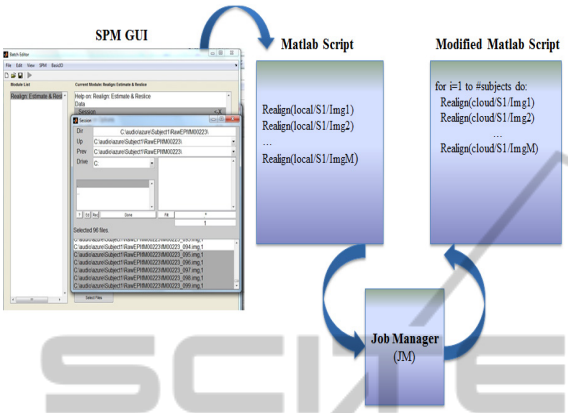

Figure 6: Process of creating SPM scripts and making them compatible with GW.

In order to store the result of the executions in the cloud storage, the user should define a name in the third argument that will be used for the output. There are some cases that do not require uploading of data when the clients submit jobs, for instance, in a specific scenario where job n-1 produces results that are required by the next job n as input. In such cases, the user can pass an explicit "`no_data`"-signal in the flag argument. This notifies the application to fetch the input data from the storage.

To fulfil other job management tasks, we implemented the following functions to track the status of submitted tasks, and to cancel a specific submitted job on GW when required by user:

`poll_status(id)` and `terminate_job(id)`

The first function resolves a job identifier that is returned from the `submit_job` function and periodically polls the status of a defined job. The status could be: Pending, Running, Finished or Cancelled. Furthermore, the client is also able to terminate any job using its job identifier.

These two additional commands are very useful when an SPM user needs to submit a large number of jobs in parallel together on different running instances of the GW. All the submitted jobs can easily be tracked, and terminated instantly if needed, using these commands.

4.4 Data Management

In order to upload, download or delete brain images on the cloud storage, we developed a set of functions to enable SPM users to run those functions from the MATLAB command line. In the first step, the DM acquires a container reference from the storage using a GW API according to the container's name:

```
GetContainerReference(container)
```

The argument of this function refers to the container that the user provided to invoke the data transfer functions. We also increased the default time-out of the client (by setting the TimeSpan to a reasonable value) so that large amounts of data could be transferred successfully even if there were delays in the underlying communication networks or storage server timeouts.

To upload a file (that is, a set of brain images as a single file) to a specific container, the client should issue the command:

```
upload_subject(container, file_name,
                local_path)
```

Moreover, to facilitate the distributed access of data for a set of results, the client can query the container that stores a particular file. For this purpose, user needs the file name that is stored in the container and the local path to save the result. The user must define at least a 512-bits storage access key as primary key, based on Azure management portal in combination with the storage account name to authorize user requests to storage services:

```
download_subject(container, file_name,
                local_path)
```

SPM users can issue another command to remove the storage contents associated with file names in distinguished containers. We implemented the following function with the same interaction pattern, with upload/download provided by the GW API:

```
delete_subject(container_name,
                file_name)
```

5 CONCLUSIONS AND FUTURE WORK

In conclusion, cloud computing as an enabling technology helps eliminating the barriers such as restricted resource scalability imposed on researchers by the existing systems in the brain imaging area. Our prototype demonstrated that cloud computing, and specifically Microsoft Azure, can

enable neuroscientists to exploit the benefits of cloud computing. Users and organizations no longer need to be concerned about hardware requirements or storage capacities as the cloud provides scalability on demand. Also, our implementation shows that it is feasible to share data securely between users belonging to different organizations.

However, while implementing our solution, we faced compatibility issues with the GW SDK and the MATLAB environment, and we had to adapt the .NET API according to the SPM settings. We were also interested in integrating the Cloud Data Management Interface (CDMI) implementation (Livenson and Laure, 2011) for transferring our data to the cloud, but it was not feasible with our solution since the CDMI implementation on Azure storage services missed some features and the .NET CDMI client library was not as functional as the Java version. Therefore we had to opt for a solution that transferred our granular files as a single unit to the cloud storage.

As we mentioned earlier in section 2.1, each SPM job is composed of several steps and in real life it would be interesting to measure the overheads associated with using Microsoft Azure for deploying the SPM. For instance, a thorough performance analysis could be carried out to compare the stage-in/stage-out time for data, the length of time that jobs stay in the queue before being run, and other metrics such as the execution or completion time for jobs. Also, it could be useful to run real SPM jobs in different VMs with different memory capacities and CPU cores to discover the limitations that our prototype might pose in real life. The possibility of designing parallel jobs through breaking the SPM jobs into smaller tasks that can be run in parallel also can be interesting to achieve better performance. Finally, to empower users with browsing and data management capabilities of the storage, implementing a MATLAB GUI would be useful. This feature will enable users to see contents of the containers that belong to different research groups for data sharing purposes.

ACKNOWLEDGEMENTS

We gratefully acknowledge the help of those who organized, reviewed and provided motivation for this paper, especially: Fredrik Ullén, Lars Forsberg, Rita Almedia from Karolinska Institute and Åke Edlund, Ilja Livenson from the Royal Institute of Technology. Special thanks also to Genet Edmondson for language comments. This research was sponsored by the EU within the 7FP project "VENUS-C" under grant agreement number 261565.

REFERENCES

Beno. Retrieved April 7, 2011, from *http://phiwave.sourceforge.net/howto_parallel/#Parallel_SPM_batch_scripting_on.*

Dean, J. and Ghemawat, S. (2004). *MapReduce: Simplified Data Processing on Large Clusters,* Sixth Symposium on Operating System Design and Implementation, San Francisco, CA.

Djordjevic, I. and Dimitrakos, T. (2005). *A Note On the Anatomy of Federation,* BT Technology Journal, Volume 23, Issue 4.

Generic Worker Complete Documentation. Retrieved June 4, 2012, from *http://resources.venus-c.eu.*

Foster, I. et al., (2007). *OGSA Basic Execution Service,* Version 1.0, GFD-RP-R-P.108.

Hwang, K., Fox, G., and Dongarra, J. (2011*). Distributed and Cloud Computing: From Parallel Processing to the Internet of Things,* Morgan Kaufmann Publishers.

Livenson, I. and Laure, E. (2011). *Towards Transparent Integration of Heterogeneous Cloud Storage Platforms,* Proceedings of the Fourth International Workshop on Data-Intensive Distributed Computing.

MCR (MATLAB Runtime Compiler). Retrieved October 24, 2011, from *http://www.mathworks.com/products/compiler.*

Microsoft Windows Azure. Retrieved July 24, 2012, from *http://www.microsoft.com/windowsazure.*

MPI (Message Passing Interface). Retrieved April 7, 2011, from *http://www.mcs.anl.gov/research/projects/mpi/.*

PSPM (Parallelized SPM). Retrieved April 7, 2011, from *http://prdownloads.sourceforge.net/parallelspm/.*

Savva, A. (Editor), (2005). *Job Submission Description Language (JSDL) Specification. Version 1.0.*

SPM (Statistical Parametric Mapping). Retrieved April 7, 2011, from *http://www.fil.ion.ucl.ac.uk/spm/.*

VENUS-C Deliverable D6.1, (2011). Report on Architecture, *http://www.venus-c.eu.*

VENUS-C FP7 Project, (2010). Grant Agreement No. 261565, *http://www.venus-c.eu.*