Beyond Grid Portals and Towards SaaS
A New Access Model for Computational Grids, in the dMRI Brain Context

Tarik Zakaria Benmerar, Mouloud Kachouane, Fatima Oulebsir-Boumghar
ParIMed Team, LRPE, USTHB, BP32, El Alia, Bab-Ezzouar, Algiers, Algeria
tarik.benmerar@acigna.com, mouloud.kachouane@gmail.com, fboumghar@usthb.dz

Keywords: Grid Computing, Grid Portals, Cloud Computing, SaaS, OAuth, MyProxy, Grid-Cloud Interoperability, dMRI Brain, Web Worker, Canvas2D, WebGL, MarionetteJS, BackboneJS.

Abstract: Acigna-G is an ongoing research project to develop a new hybrid Grid SaaS architecture. CloudMRI is our proof-of-concept Acigna-G based SaaS Service for the Brain dMRI field. The main objective of such architecture is to provide local (Browser) and remote (Grid) intensive computational capabilities completly abstracted to the SaaS user. The result is a combination of an in Browser Rendering and Computation engine, interoperable REST-SOAP Grid Services, and interoperable web-grid authentication mechanisms. Such architecture can allow new types of SaaS Services, specifically for the dMRI Brain field.

1 INTRODUCTION

At the higher level of the Cloud Computing architecture, SaaS Services provide ubiquitous access to hosted software stack. These softwares provide an intuitive and an interactive access to user's Cloud Data.

From the computational standpoint, we have observed that the current SaaS Services do not provide the intensive computational capabilities, while remaining highly interactive and intuitive, with rich 2D/3D rendering.

Also, the grid computational capabilities have not been used in conjunction with SaaS Cloud Services, to abstract the underlying job submission mechanisms.

The current grid portals use somewhat intuitive web interfaces, but still follow the job submission paradigm, where the user is provided with job submission forms and basic visualization tools.

Through the ongoing Acigna-G project and platform, we have investigated a hybrid SaaS Cloud Grid architecture. The SaaS Cloud Service provides a rich interactive and intuitive web interface, with in browser 2D/3D rendering and basic computations. The intensive computational part of the service are delegated to the grid infrastructure, while the user is nearly unaware of this fact.

This architecture has been developed under the umbrella of the dMRI Brain field. CloudMRI is the resulting SaaS Service built with this architecture.

The present paper explains the importance of such architecture. Its remaining parts are structured as follows. In Section 2, we introduce the dMRI Brain Tractography and related softwares. In Section 3, we explain the importance of a Hybrid SaaS Grid Architecture. In Section 4, we present the Acigna-G Platform and some parts of the its built upon CloudMRI SaaS Service. In Section 5, we present a comparative study. In Section 6, we conclude the paper.

2 dMRI BRAIN TRACTOGRAPHY

With the Diffusion-weighted imaging (DWI), we can investigate a tissue microstructure in vivo and noninvasively. By combining MRI diffusion with imaging DWI became possible, enabling the mapping of both diffusion constants and diffusion anisotropy inside the brain and revealing valuable information about axonal architectures (Le Bihan and Breton, 1985).

Later, describing the degree of anisotropy and the structural orientation information quantitatively became possible with the introduction of Diffusion Tensor model (Basser et al., 1994). With only six parameters, this Diffusion Tensor Imaging (DTI) was able to provide a simple and elegant way to model this complex neuroanatomical information.

Different regions of the brain are connected through axons, that form the brain white matter. White matter tracts represent a set of axons that share a similar destination. With tractography or fiber-tracking algorithms, major tracts can be clearly ob-

served by DTI with 2-3 mm image (Tournier et al., 2011) (See Figure 1).

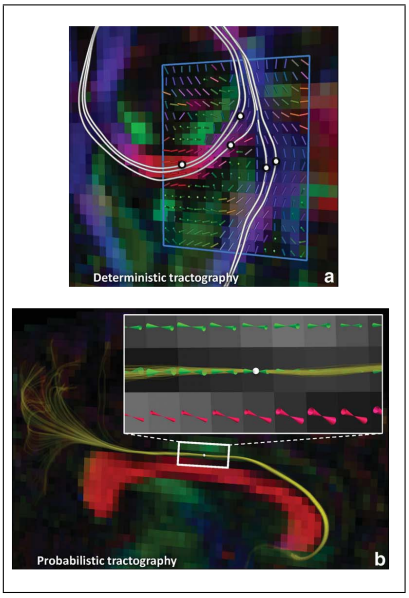

Figure 1: Deterministic VS Probabilistic tractography (Tournier et al., 2011).

Differing in quality and performance, there are mainly two types of tractography:

- **Deterministic Tractography:** This tractography supposes that from a starting point or seed, there exists exactly one tract that goes out from it (See a in Figure 1). This algorithm is fairly fast (a few minutes to less than a hour), but produces limited results.
- **Probablistic Tractography:** This tractography supposes that from a starting point or seed, there can be many tracts that go out from it (See b in Figure 1). This algorithm is heavy in processing (several hours), but produces better results.

From the dMRI Brain Software standpoint, MedInria (MedInria, 2014) provides an intuitive and an interactive desktop application. Neverthless, it is fairly limited as it provides only deterministic tractography, due to the limited ressources of a PC.

On the other hand, FSL (FSL, 2014) provides access to advanced probablistic tractography. But, access to such algorithms are through a non intuitive command line, as it is required for a grid deployment.

3 HYBRID SaaS GRID ARCHITECTURE

From the dMRI Brain Tractography algorithms and related softwares introduced in the previous section, we argue that the existance of a hybrid SaaS Grid architecture is very important for the dMRI Brain field.

In fact, the current SaaS Services such as Google Docs (Google, 2014), provide a visualization and a manipulation tool to act on your Cloud data. These services provide the same features as their desktop counterparts, with the additionnal ubiquitous access to Cloud Data hosted on the Cloud Provider infrastructure.

Neverthless, we have seen that the computational axis of the software is nearly absent in the current services. Hence, the Cloud Providers provide only data hosting and manipulation, and no computational capabilities are provided through the software abstraction level.

In the Grid context, Grid Portals can provide all the computational capabilities required. With a web interface, the user can submit jobs and visualize the end result inside a web browser.

Through the recent years, Grid Portals have tried to cope with the evolution of the web, specifically the web 2.0 trend (Pierce et al., 2006). RSS feeds, social networks, wikis and visualization through interactive maps, have been incorporated.

Neverthless, advanced interaction and 2D/3D rendering that we find in SaaS services haven't been integrated. Also, the job submission paradigm is still the only paradigm followed, in the sense that a Grid Portal is all about job submission and result visualization. No advanced software abstraction exists, where the user is totally unaware about any job submission.

Through our Acigna-G initiative (Benmerar and Oulebsir-Boumghar, 2011) (Benmerar and Oulebsir-Boumghar, 2012), we have tried to provide such a hybrid architecture after issues faced with a previous grid plaform (Oukfif and Oulebsir-Boumghar, 2009). Through iterative improvements, the architecture has evolved from a platform for Linux shell-like interactive web access to computational tasks, to the actual SaaS service completly abstracted from the underlying grid infrastructure.

CloudMRI is our proof of concept, where we use the Acigna-G platform to deliver a dMRI Brain SaaS. It allows to provide the same interactive experience as MedInria, with the additional access to advanced processings such as probabilistic tractography, implemented in FSL.

4 THE ARCHITECTURE OF THE ACIGNA-G PLATFORM AND THE CLOUDMRI SaaS SERVICE

In this section, we present the Acigna-G architecture and its built upon CloudMRI SaaS Service.

The architecture is composed into four parts: In Browser Computing and Rendering, Restful Grid Services, Grid-Cloud Interoperable Authentication and Computing Grid Services. We introduce each part in the following subsections:

4.1 In Browser Computing and Rendering

The CloudMRI web frontend has been built as a Single Page Application (or SPA). Rather than decomposing the application into multiple pages, it is hosted under one url that dynamically fetches the required ressources using AJAX, then emulates passing from one url to another. We usually find this technology in modern web mail clients, we use daily. MarionetteJS (Bailey, 2014) and BackboneJS (DocumentCloud, 2014) have been used as the underlying technology for this purpose. It was required to build the software in this way, as we wanted a high interactivity at the frontend.

The frontend uses the Acigna-G MyThreadJS Javascript library, to manage computations, rendering and data sources. These different axis of the library are explained in the following paragraphs:

Computational Management

The Compute Manager Component provides a central entry point to local and remote job execution. The remote jobs are delegated to the Restful Grid Service presented in 4.2.

Locally, the Compute Manager manages threads that execute either in the local browser thread, or in a separated thread using Web Workers standard (Mozilla, 2014b).

The local threads are managed in a way that avoid blocking user browser interaction. They are built with interruption points specified by the thread developer, that the compute manager uses to schedule threads execution.

Currently, our CloudMRI frontend uses particularly the Computer Manager for parsing NIfti (NIMH, 2014) file format, through a threaded class called ParserNII. This processing can be heavyweight, and the component helped avoid any threading issues.

Rendering Management

The Render Manager Component constitutes a central point for efficient scheduled rendering. Rather than immediatly rendering after a user interaction, it schedules it at a specific time interval to avoid costly reprocessing which can be the case of 2D/3D rendering using Canvas2D (Mozilla, 2014a) or WebGL (Khronos, 2014) standards respectively.

Currently, the CloudMRI fronted uses the Render Manager to render MRI Slices with a component called MRISliceWidget, illustrated in Figure 2.

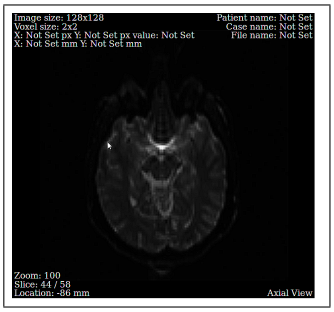

Figure 2: MRI Slice Widget.

Data Source Management

The Source Manager component in myThreadJS library provides a central entry point to local and remote data fetching, caching, persisting and locking to the rest of the application. Two types of data are managed by this component: local data and remote data.

Local data are data hold on the user file system, that stay persistent across sessions. Remote data can be considered as cloud data. The Indexed Database API (W3C, 2014) is used for persisting local data, and for caching remote data.

The data sources are referenced across the application with a point separated string (e.g: 'patient.mri'). The point separation is used to differentiate data namespaces, or for building a hierarchy of data sources processings (e.g 'patient.mri.parsed' has been generated from the 'patient.mri' processing). Such hierarchy is used to avoid later reprocessing of data.

To avoid that two threads process the same data, locking data sources mechanisms exist. For example, if the first thread is processing 'patient.mri' to generate 'patient.mri.parsed', the later is marked as locked. The second thread waits for the availablity of the later data source, after the first thread processed it and unlocked it.

The CloudMRI frontend uses the Source Manager for the user's Brain dMRI Images persistance.

4.2 Restful Grid Service

To submit a job from the SaaS Front-end, we have designed two types of Restful services. They are intermediaries between the SaaS Front-end and the Computing Grid Services. Their role is to avoid the construction of a heavy weight XML SOAP envelope at the browser side, by using JSON data format.

Restful BES Service

Restful BES Service is a REST equivalent to the OGSA-BES (OGF, 2008) standard. This service is based on the proposed solution in (Andreozzi and Marzolla, 2009), but uses JSON for the JSDL document (OGF, 2005). The JSON format construction follows particular rules, that are illustrated in Figure 3 for sample examples.

Figure 3: Conversion examples between JSDL JSON and JSDL XML, for sample fragments.

Task Specification Restful Service

The Task Specification Restful Service avoids the construction of a whole JSDL document, by just requiring a task name and parameters. The Restfull Service retrieves the corresponding JSDL document and inserts the specified parameters. This service can be important when the SaaS service has a predefined set of jobs that can be submitted.

4.3 Grid-Cloud Interoperable Authentication

In the proposed architecture, we have started from the hypothesis that the Cloud Provider is an independant third-party, separated from the Computing Grid. The SaaS Provider should authenticate itself to the grid, before job submission. We have then supposed two different scenarios: Grid User and Non-Grid User, explained in the following paragraphs.

Grid User

In such scenario, the SaaS Service provides computational capabilities of the grid infrastructure the user has access to. The Service Provider should then have access to the user credentials to authenticate itself. These credentials are usually stored in a MyProxy Credential Management Service and protected with a login and a password.

Providing such information (login and password) to the Service Provider is not secure, as it could lead to user's credentials abuse by the provider. A solution proposed in (Basney et al., 2011) for grid portals, avoids enveiling the login and password. The architecture uses an OAuth service in front of the MyProxy service, as depicted in Figure 4.

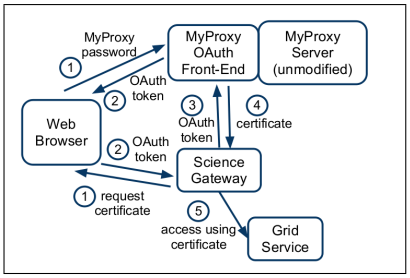

Figure 4: Science Gateways Using OAuth with MyProxy (Basney et al., 2011).

Using OAuth tokens, the user provides access to his MyProxy account, without enveiling his login and password. At a later time, these tokens can be revoked and the third-party will no longer have access to the MyProxy account.

Without change, such solution can be used in our architecture as a means to provide the required computational capabilities to grid users.

Non-Grid User

In the case of a non-grid user, other mechanisms are required for the Service Provider to be allowed to submit jobs to the underlying grid infrastructure.

Dynamic Accounts mechanism (Insley et al., 2009) used in the grid portal context, can be integrated in our architecture. With Dynamic Accounts, the computing grid can allocate dynamically a pool of user accounts to the grid portal. These accounts can be mapped to the real users at runtime.

In the SaaS context, the Service Provider could buy a pool of dynamic accounts from the computing grid. It could then use them to provide the required computational capabilities to the SaaS users.

4.4 Computing Grid Services

At the Grid level of our architecture, a set of Python-based grid services are deployed. These services have been developped using the ZSI (ZSI, 2013) Python library.

The ZSI library is a python library for web services deployment and use. We have successfully deployed a basic WS-BES Grid Service that accepts a JSDL template for the job submitted.

But different extensions such as JSDL SPMD (Savva, 2005) are required for parallel tasks deployment. We are currently working on such extensions, to provide a more complete capabilities at the grid service level.

We have privilegied the use of Python, as we have found it easier for deployment than for example Java based services. There is for example no need for precompilation before deployment. Furthermore, Python-based webservices exhibit comparable performances to Java-based ones (Santos and Koblitz, 2005).

Neverthless, the deployment required a bit of patching on the source code, and we are planning to support the open source project through source code contributions.

HTCondor (HTCondor, 2014) is used as the job submission system at the grid site level. Though other job submissions systems can be used, an adaptor plugin should be developped for each one, at this moment.

5 COMPARATIVE STUDY

NEWT (Cholia et al., 2010) is a RESTFul Service provided by NERSC for building High Performance Computing Web Applications. It provides an easy API for authentication, persistance management and job submission. It uses the lightweight JSON data format for data-exchange between the client application and service.

Neverthless, the service is tightly-coupled to the underlying NERSC infrastracture. When submitting a job, a specific machine queue should be specified. In some cases, the user should even specify the required batch system script (e.g for PBS).

Also, the application examples using this platform, still follows the job submission paradigm we have presented. They don't provide an interactive software environment, abstracted from the infrastructure detail.

The BES-REST (Andreozzi and Marzolla, 2009) exhibits the required service abstraction by providing a Restful interface equivalent to an established Grid Job Submission Standard (WS-BES).

This work has been the basis of our Restful BES Service, with the difference that we use the lightweight JSON data-format rather than XML for the JSDL job specification. This was required for an efficient job submission from the browser.

Cantor et al. (Cantor-Rivera and Peters, 2010) have proposed a web application architecture for remote MRI rendering, using WebGL. The architecture doesn't cover any local or remote computional capabilities.

XTK (XTK, 2014) is a WebGL toolkit, for Scientific Visualization. With XTK, medical images formats can be rendered in the browser, such as NIfTI format.

Compared to our work, XTK neglects threading issues that we have solved with the MyThreadJS library. This constitutes a problem in a heavy software environment.

In (Rings and Grabowski, 2012), the authors investigate the integration of Cloud and Grid infrastructures. The architecture tries to integrate computational ressources situated on an IaaS Cloud or a Computational Grid. It differs significatly from our work, as we have looked into SaaS (rather than IaaS) Clouds and Grids integration.

In (Church et al., 2012), the authors investigate the deployment of an HPC environment in a SaaS Cloud. All the computational ressources are provided by an underlying IaaS provider. The SaaS Service provides a web interface to submit computational jobs in the Cloud Infrastructure. Unlike our architecture, the service follows the job submission paradigm and don't manage local computations.

6 CONCLUSION

Current SaaS Services can evolve to provide computational capabilities, beyond the data capabilities actually provided. On the other hand, Grid Portals can evolve to provide an interactive software environment completly abstracted from the enderlying grid infrastructure, that would be easier and intuitive to use.

The proposed architecture tries to fill in these observed gaps. Particulary, we have emphasied the need of such architecture in the Brain dMRI Field.

Currently, much of the functionalities of the myThreadJS library have been developed and tested. The results have been very satisfying with respect to performance and interface responsiveness, and a indepth comparaison with the XTK library is under preparation. We have also developed and tested the

submission of a grid job into HTCondor through a basic Restful BES and WS-BES Grid Service.

Other parts of Acigna-G are still under development. We have also began the development of the CloudMRI SaaS Service with the first basic user interfaces being setup.

In subsequent papers, we will provide detailed reviews of the architecture and its performances. This is necessary as we are heading away from a developping platform to a testing, and then a production one. This will be characterised by rich feedbacks from the concerned users, particulary our research colleagues.

REFERENCES

Andreozzi, S. and Marzolla, M. (2009). A restful approach to the ogsa basic execution service specification. In *Proceedings of the fourth International Conference on Internet and Web Applications and Services ICIW '09*.

Bailey, D. (2014). Marionettejs website. Retrieved January 13, 2014. http://www.marionettejs.com.

Basney, J., Dooley, R., Gaynor, J., Marru, S., and Pierce, M. (2011). Distributed web security for science gateways. In *Proceedings of the 2011 ACM workshop on Gateway computing environments GCE '11*.

Basser, P., Mattiello, J., and Le Bihan, D. (1994). Mri diffusion tensor spectroscopy and imaging. *Biophysical Journal*.

Benmerar, T. Z. and Oulebsir-Boumghar, F. (2011). Toward a cloud architecture for medical grid applications : The acigna-g project. In *Proceedings of the 10th International Symposium on Programming and Languages ISPS '2011*.

Benmerar, T. Z. and Oulebsir-Boumghar, F. (2012). From grids to cloud : The pathway for brain dmri cloud services. In *Proceedings of the 2nd International Conference on Cloud Computing and Services Science CLOSER '2012*.

Cantor-Rivera, D. and Peters, T. (2010). Pervasive medical imaging applications : current challenges and possible alternatives. Retrieved January 13, 2014. http://imaging.robarts.ca/ dcantor/wp-content/uploads/2010/07/eHealth.pdf.

Cholia, S., Skinner, D., and Boverhof, J. (2010). Newt: A restful service for building high performance computing web applications. In *Proceedings of the Gateway Computing Environments Workshop (GCE)*.

Church, P., Wong, A., Brock, M., and Goscinski, A. (2012). Toward exposing and accessing hpc applications in a saas cloud. In *Proceedings of the 2012 IEEE Nineteenth International Conference on Web Services ICWS '2012*.

DocumentCloud (2014). Backbonejs website. Retrieved January 13, 2014. http://www.backbonejs.org.

FSL (2014). Fsl website. Retrieved January 12, 2014. http://www.fmrib.ox.ac.uk/fsl/.

Google (2014). Google docs. Retrieved January 10, 2014. http://docs.google.com.

HTCondor (2014). Htcondor website. Retrieved January 13, 2014. http://research.cs.wisc.edu/htcondor/.

Insley, J. A., Leggett, T., and Papka, M. E. (2009). Using dynamic accounts to enable access to advanced resources through science gateways. In *Proceedings of the 5th Grid Computing Environments Workshop GCE '09*.

Khronos (2014). Webgl website. Retrieved January 13, 2014. http://www.khronos.org/webgl/.

Le Bihan, D. and Breton, E. (1985). Imagerie de diffusion in-vivo par résonance magnétique nucléaire. *Comptes-Rendus de l'Académie des Sciences*.

MedInria (2014). Mediniria website. Retrieved January 12, 2014. http://med.iniria.fr.

Mozilla (2014a). Canvas. Retrieved January 13, 2014. http://developer.mozilla.org/fr/docs/Web/HTML/.

Mozilla (2014b). Using web workers. Retrieved January 22, 2014. http://developers.mozilla.org/en-US/docs/DOM/Using_web_workers.

NIMH (2014). Nifti format. Retrieved January 13, 2014. http://nifti.nimh.nih.gov/nifti-1/.

OGF (2005). Job submission description language (jsdl) specification. Retrieved January 10, 2014. http://www.ogf.org/documents/GFD.56.pdf.

OGF (2008). Ogsa basic execution service (ogsa-bes) version 1.0 specification. Retrieved January 10, 2014. http://www.ogf.org/documents/GFD.108.pdf.

Oukfif, K. and Oulebsir-Boumghar, F. (2009). Prétraitement parallèle de qsplat sous grille de calcul. In *Journées Scientifiques sur l'Informatique et ses Applications*.

Pierce, M. E., Fox, G., Yuan, H., and Deng, Y. (2006). *Cyberinfrastructure and Web 2.0.*, volume High Performance Computing and Grids in Action. IOS Press.

Rings, T. and Grabowski, J. (2012). Pragmatic integration of cloud and grid computing infrastructures. In *Proceedings of the 2012 IEEE fifth International Conference on Cloud Computing CLOUD '2012*.

Santos, N. and Koblitz, B. (2005). Metadata services on the grid. *Nuclear Instruments and Methods in Physics Research*.

Savva, A. (2005). Jsdl spmd application extension, version 1.0 (draft 008). Retrieved January 10, 2014. http://www.ogf.org/Public_Comment_Docs/Documents/May-2007/draft-ogf-jsdl-spmd-008.pdf.

Tournier, J.-D., Mori, S., and Leemans, A. (2011). Diffusion tensor imaging and beyond. *Magnetic Resonance in Medicine*, 65:1532–1556.

W3C (2014). Indexed database api. Retrieved January 13, 2014. http://www.w3.org/TR/IndexedDB/.

XTK (2014). Xtk website. Retrieved January 13, 2014. http://www.goxtk.com.

ZSI (2013). Zsi website. Retrieved January 10, 2014. http://pywebsvcs.sourceforge.net/zsi.html.