Nekkloud: A Software Environment for High-order Finite Element Analysis on Clusters and Clouds

Jeremy Cohen*, David Moxey†, Chris Cantwell‡, Pavel Burovskiy†, John Darlington*, Spencer J. Sherwin†
*Department of Computing, Imperial College London, South Kensington Campus, London SW7 2AZ, UK
Email: jeremy.cohen@imperial.ac.uk
†Department of Aeronautics, Imperial College London, South Kensington Campus, London SW7 2AZ, UK
‡National Heart & Lung Institute, Imperial College London, South Kensington Campus, London SW7 2AZ, UK

***Abstract*—As the capabilities and diversity of computational platforms continue to grow, scientific software is becoming ever more complex in order to target resources effectively. In the libhpc project we are developing a suite of tools and services to simplify job description and execution on heterogeneous infrastructures. This paper presents Nekkloud, a web-based software environment, built on aspects of the libhpc framework, for running the Nektar++ high-order finite element code on both cluster and cloud platforms, while improving the accessibility of the software for end-users and improving the user experience. Nektar++ provides a suite of solvers which span a range of scientific domains, ensuring that Nekkloud has a broad range of use cases. We describe the Nekkloud environment, its use and its ability to target both local campus cluster infrastructure and cloud computing resources, enabling users to make better use of the facilities available to them.**

I. Introduction

As computer architectures have evolved, per-processor core counts have increased but maximum clock speeds have remained static. Developers have therefore been forced to efficiently parallelise their code in order to take advantage of the additional computing power available from multi-core processors, multi-node infrastructures such as clusters and clouds and more exotic coprocessor computing platforms. For many scientists and researchers, High Performance Computing (HPC) infrastructure takes the form of local campus-based clusters. Setting up and fine-tuning a code to run efficiently on such computing resources is often a challenging task and usually involves a highly technical understanding of the system configuration and underlying hardware. Cloud computing environments provide an alternative avenue, enabling on-demand HPC capabilities, but come with an additional set of technical hurdles in terms of resource preparation and deployment.

The challenges associated with making effective use of HPC resources have been acknowledged previously in the literature [6], [18]. E-Science research has developed a range of solutions to the many challenges of building and running complex scientific applications, particularly in the context of Grid Computing infrastructure, e.g. [7], [9], [13]. Tools and services have been developed which support scientists in using large-scale software more effectively, while reducing their need to acquire complex technical computing skills, e.g. [15], [20]. With the emergence of new methods of accessing computing resources, for example, on-demand Infrastructure-as-a-Service (IaaS) cloud computing platforms (e.g. Amazon EC2 [5]), scientists have further options for utilising different numbers and types of resources and controlling how they run their code.

There has also been extensive work on the development of application portals or science gateways [21] to support the running of particular applications or groups of applications, for example [17]. Services such as Apache Rave provide toolkits to develop portals for orchestrating web-based services and data [14] while workflow management systems such as Taverna [8] offer a means to build complex workflows based on multiple external computational services. Other solutions are domain specific. For example, in computational biology, the *Galaxy CloudMan* [3] web interface provides user-friendly access to cloud infrastructure for scientists using the Galaxy platform, a suite of computational biology research tools. Similarly, the *BioHPC* project [19] presents the user with a web interface for deploying computational biology simulations on Windows HPC clusters.

In this paper we present Nekkloud which brings together the concept of an application portal with infrastructure access and management services developed as part of the libhpc framework [1]. It provides a web-based software environment to support the use of Nektar++ [2], a high-order finite element framework, on both clusters and private/public clouds by scientists and engineers across a range of application domains. The libhpc framework is being developed to simplify the execution of HPC codes in distributed, heterogeneous computing environments. It is similar in its aims to Apache Airavata [16] but differs in its use of co-ordination forms – high-level functional constructs – to describe applications, its model of interchangeable optimised component implementations and its core mapping services that dynamically select the most appropriate components and hardware based on available resources. At present Nekkloud uses only the libhpc deployment service and in this work we have extended this service with a new connector that provides the ability to deploy to clusters running PBS Pro [4]. This has enabled the Nekkloud system to operate with the Imperial College High Performance Computing service that is managed by Imperial College for use by its researchers and academics.

We consider the main contributions of this paper to be:

- The development of a modular and extensible software environment to enable scientists and non-technical users to perform large-scale Nektar++ simulations on either cluster or cloud infrastructure.

978-1-4799-0898-1/13/$31.00

- Integration of the libhpc deployment service with both existing campus cluster resources and private/public cloud infrastructure.
- A web-frontend module for the exemplar application Nektar++ and demonstration of an improved user experience when running Nektar++ jobs.

We note that the aim of this paper is to present an overview of the Nekkloud system, our motivations for its design and the benefit of such a system for end-users. As such, we do not consider performance or security related matters in this paper. However, the performance and relative scalings on both cloud and cluster architectures is a topic that we are currently investigating in more detail in other work [11].

In section II we outline the current challenges for non-technical users in using finite element codes on HPC facilities and describe our motivation for building Nekkloud, outlining the goals of the project. In section III we provide details of the architecture and implementation of Nekkloud and give an overview of the components which are used to construct the system. We then discuss the experience for an end-user using Nekkloud in section IV and present a brief use case example.

II. Challenges & Motivation

Finite element codes are typically used by scientists who have sufficient computer science expertise to deploy and run the code on a local standalone system or on a homogeneous compute cluster provided by their institution or enterprise. Consequently, users lacking this expertise are unable to run such codes without support from a computer scientist or someone with specialist computing knowledge. Even for technically trained users, modern HPC facilities are becoming increasingly complex and difficult to use, especially if maximum performance is to be attained. In addition to the challenges developers face in writing and building code to make effective use of target resources, end-users need to deploy this code on their resources which will include installing binaries or building from source. Typically, modern HPC software also requires the installation of multiple dependencies, which further complicates the compilation process. Finally, to run a job on a cluster, users will often need to write shell scripts or otherwise define job-specific information, and additionally transfer files to and from the cluster manually.

Significant manpower is therefore required in order to train users to effectively use cluster resources, and moreover the lack of technical knowledge is often too large for many users to contemplate using HPC resources. Additionally, users will only tend to work with a small number of clusters (usually local systems and possibly a national facility). This discourages users from diversifying their infrastructure usage, even when other potentially more powerful resources are available.

It is also noteworthy that scientists and engineers who would like to run large scale simulations do not always have access to appropriate resources. The financial costs of installation and maintenance of a large local cluster are often too high, and in many cases there is not sufficient demand inside an institution to warrant such expenditure. In these cases, the use of cloud computing resources and remote clusters provides an attractive alternative and gives the end-user a range of computing facilities to suit a wide variety of cost and resource requirements. Even where cluster access is available, queueing times on local resources may mean that for time-critical applications, cloud access is more desirable as resources are usually instantly available through large public-domain providers. However, for many end-users, the technical challenge of building custom images for virtual machines, which are prevalent throughout the cloud environment, precludes their use of this type of resource.

The ultimate goal of the Nekkloud system is therefore to address these issues by providing a flexible and easy-to-use interface which encapsulates many of the difficulties users may have when deploying to different types of resources. We focus, therefore, on a single exemplar application – the Nektar++ framework – which has applications in many scientific fields. The design of the system means that end-users without extensive computer science knowledge are able to undertake Nektar++ computations using the platform that most suits their requirements, at the same time isolating them from the intricate details which are involved in deployment for each platform.

In this paper, we describe how Nekkloud can provide an effective interface to real-world scientific applications through a flexible deployment layer. Nektar++ itself is a powerful open-source framework written in C++ for solving partial differential equations (PDEs) from a variety of scientific domains using the spectral/*hp* element method, an extension of the linear finite element method (FEM) to use high-order polynomials. We describe an example use case based on the Nektar++ cardiac electrophysiology solver to show how Nektar++ can be effectively harnessed by non-technical users in order to provide real-time patient-specific feedback for clinicians through the Nekkloud interface.

To conclude this section, we summarise below the main motivations and aims in the design of Nekkloud:

- Minimise the computer science training needed by scientists to take advantage of HPC facilities.
- Enable seamless access to both campus HPC infrastructure as well as private and public cloud platforms.
- Encourage a more efficient use of available computing infrastructure by allowing users to switch easily between resources.
- Present a simple, high-level user interface for job specification that removes the need for command-line tools.

III. Architecture & Implementation

The user-facing element of the Nekkloud system is a web-based interface that is designed specifically to support the solvers provided by Nektar++ and the properties required by these solvers. The back-end of the system uses components from the libhpc framework handling the software deployment and execution. While Nekkloud provides an interface targeted at Nektar++, alternative web interfaces could be developed to support different applications or domains while continuing to use the same back-end libhpc services.

The Nekkloud web application is implemented in Python using the Django Web framework. Web pages are built using the Bootstrap CSS and Javascript framework. The

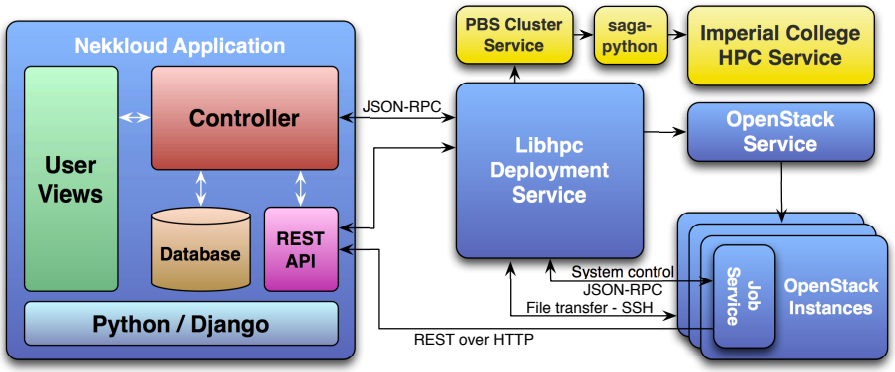

Fig. 1. Architecture of the Nekkloud system.

web application uses the libhpc deployment service's `JobServiceClient` class which communicates with the server-side libhpc deployment service using JSON-RPC communication via Python's jsonrpclib library [12]. The deployment service provides connectors that allow it to support different types of underlying hardware platforms such as Portable Batch System (PBS) clusters and private/public clouds.

A. Nekkloud System Components

Figure 1 shows the architecture of the Nekkloud system which consists of the main Nekkloud application and web-based user interface which makes use of our existing libhpc deployment service along with its OpenStack private cloud connector service and the PBS Cluster service used to target PBS-based clusters. We now provide a brief description of the core components of the Nekkloud system:

Controller: The controller is the core of the system and handles the main application logic. It works with data from the local database, data that has been provided by other libhpc services via the REST API or data that comes from users via the web views.

Database: The database stores user, job and input/output file information.

User views: Provide the Nekkloud web pages that users interact with via their web browser.

REST API: The REST API is designed to be accessed by other internal libhpc services that need to provide information to, or obtain information from, Nekkloud (e.g. updating job status information).

Libhpc Deployment Service: The deployment service handles the configuration and execution of jobs on remote resources. Connectors allow it to work with a range of underlying infrastructures such as cluster batch submission systems and infrastructure clouds. Details of the existing libhpc deployment service and the OpenStack connector are described in [11].

PBS Cluster Service: The PBS cluster service leverages saga-python [20] to allow deployment of MPI-based parallel jobs on PBS-based clusters.

OpenStack Service: The OpenStack service provisions infrastructure cloud resources on our local, private OpenStack cloud, configures MPI and then submits and runs parallel jobs on these resources.

B. PBS Cluster Service

The PBS Cluster Service is a deployment connector that has been developed for the libhpc deployment service to enable the running of jobs on clusters managed by PBS. The connector was developed to allow the integration of the central Imperial College HPC service with the libhpc framework and, ultimately, for this platform to become a target for running jobs from Nekkloud. The service makes use of the saga-python library that provides a Python implementation of the Simple API for Grid Applications (SAGA) [10]. Saga-python provides a series of adapters for connecting to and interacting with remote resources or job submission systems. We use the PBS SSH adapter to allow us to connect remotely to the interactive job submission node for the target cluster and submit and monitor jobs.

In order to run a user's job on a PBS cluster, Nekkloud communicates with the libhpc deployment service passing the job specification, including the required deployment resources, as entered by the user via Nekkloud's web interface. The deployment service delegates control to the relevant platform connector, in this case the PBS Cluster Service, which initialises and then runs the job.

When the job runs on the PBS cluster it will generate output data that needs to be stored. When initialising a new job, the PBS connector sets up a job directory on the target cluster using saga-python's file and directory tools. Input data files will have been uploaded by Nekkloud to a temporary staging area on the application server and they are then copied from here to the job directory on the remote cluster using saga-python. A SAGA Job object is then created and configured with the job properties. Saga-python generates a PBS job script based on the properties of this Job object. For our local HPC service jobs it is necessary to run some commands before and after the main MPI parallel job is executed. The SAGA Description object, which is used to provide the properties to initialise a new Job object, did not offer the ability to add pre- and post-execution commands to the script generated by saga-python, so a small modification was made to allow this. The pre-execution commands are required to initialise some properties of the cluster environment before an MPI job can begin running and

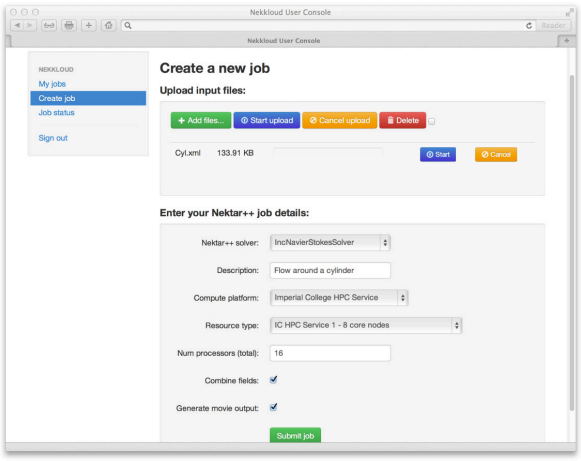

Fig. 2. The Nekkloud job creation page.

the post-execution commands are used to compress output data into an archive file to be sent back to the application server.

The PBS Cluster Service is implemented as a command-line tool to which a set of parameters are provided. The libhpc deployment service runs an instance of the PBS deployer in a new process every time a job is to be run. This avoids potential threading problems in Python that would occur if multiple instances of the PBS deployer were run in separate threads within the main libhpc deployer application.

IV. User Experience

The user facing element of Nekkloud is a web-based interface through which users specify and manage their jobs. The target audience for the user interface is scientists who may not have a computing background and the aim in designing it has been to ensure that it is uncluttered and straightforward to use. Once an account has been set up, users can log in to the system and configure and run their Nektar++ jobs. A job consists of details of the Nektar++ solver to be used, the input data file(s), hardware requirements and possibly other job-specific information that may be available depending on the solver the user has selected (see Figure 2). Nektar++ input files are in XML format and end users of the software are likely to either be familiar with creating these files themselves, or understand enough about the file content to make necessary changes to a template file provided by an experienced user. The platform selection box allows the user to select between local cluster resources provided by the Imperial College HPC service, a local OpenStack private cloud infrastructure and resources provided by the Amazon EC2 public cloud service [5]. When a selection is made from the "Compute platform" drop-down box, the "Resource type" list is updated with the types of resources provided by the selected platform. Based on the resource type selected, the system then calculates how many compute nodes must be started to provide the total number of cores requested by the user. Figure 3 shows the resource types that are available on the Imperial College HPC service clusters when this platform is selected.

Job input files are selected and uploaded via the web interface which also provides options to undertake the post-processing tasks of combining the generated field files and

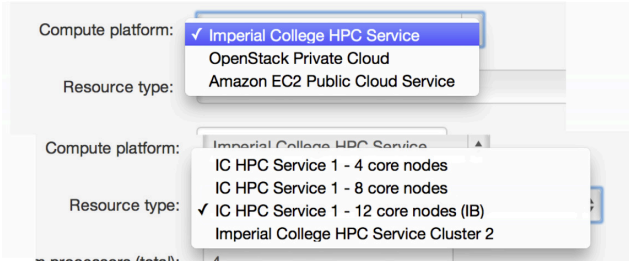

Fig. 3. The resource types provided by the IC HPC service cluster.

producing a video animation. A running job can be monitored via the job status page (see Figure 4) which displays current job status and a live feed of output data from the job execution.

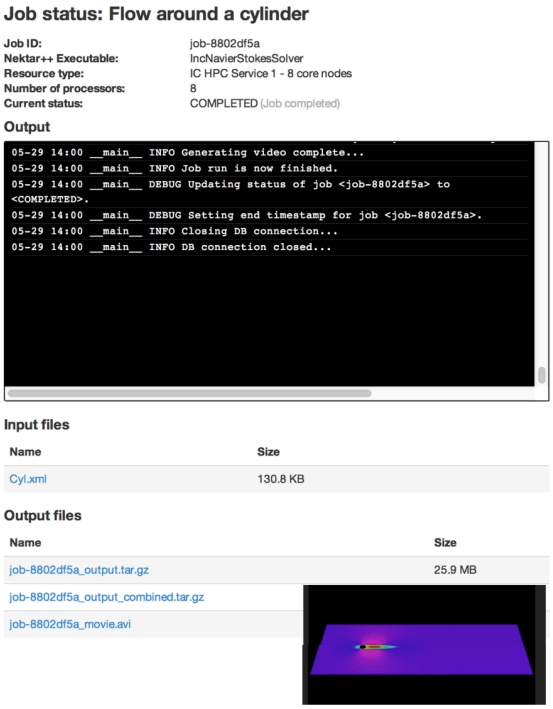

Fig. 4. Main panel of Nekkloud job status page showing debug output for a job run on the IC HPC cluster, and inset, sample output.

Nekkloud's approach contrasts with the traditional method of running Nektar++ which involves installing binaries or building source code, manually preparing job submission scripts or provisioning cloud resources. Once a job has run, the user is responsible for collecting and managing output data and writing any scripts required for post-processing output data.

A. Example use case: Cardiac Electrophysiology

Our use case example aims to highlight the benefits of the Nekkloud web interface and how it can enable users who are not equipped with specialist HPC knowledge to run finite element codes on HPC resources such as clusters and clouds. We consider a cardiac electrophysiologist, who is performing cardiac ablation on a patient with an atrial arrhythmia. Prior to the procedure the patient has been imaged using Magnetic Resonance Imaging (MRI) and a geometry and finite element

mesh has been constructed of the patient's left atrium using third-party tools. During the procedure the electrophysiologist gathers electrical recordings from inside the patient's heart. To complement this static information, the electrophysiologist can use the finite element model to simulate the electrical propagation patterns through the patient-specific atrium model. The electrophysiologist uses Nekkloud to launch a simulation, based on Nektar++'s Cardiac Electrophysiology solver, of the patient's left atrium, providing the necessary electrophysiological parameters on the web form and targeting the simulation at 1024 cores of the institution's local compute cluster. In the event that the cluster is busy, the clinician might choose to pay to run the job immediately on public cloud infrastructure. After a few minutes the job completes and the electrophysiologist views the generated video of the simulation to provide insight into the next step of the clinical procedure. The job would have taken many hours and may have required specialist computing support to run on a desktop system in the clinical environment.

V. Conclusions

We have presented Nekkloud, a web application that enables the running of the Nektar++ high-order finite element analysis software on cluster and cloud infrastructure. Nekkloud offers a significantly improved user experience, when compared to the traditional command-line execution approach, for the scientists and researchers that run the various solvers that Nektar++ provides. Nekkloud plugs in to our existing campus clusters, other remote PBS-based clusters, or cloud platforms, allowing the running of more jobs at once or larger jobs than may be possible with local infrastructure. Nekkloud is currently a prototype and we have focused on the benefits it can offer to end users with limited HPC knowledge by providing a new approach to running Nektar++ rather than looking at the performance of running jobs through Nekkloud. However, tests running Nektar++ on more than 1,000 cores have shown that the library scales well. While the raw performance of running computations on cloud infrastructure without high-performance interconnects will be lower than on a dedicated HPC cluster, the ability to easily scale computations to larger numbers of cores on clouds helps to mitigate such differences. In the future it is intended to extend Nekkloud with automatic platform selection based on user-specified metrics such as cost or time, which is a planned addition to the libhpc framework.

Acknowledgment

The authors acknowledge the support of members of the Nektar++ development team and would like to thank the UK Engineering and Physical Sciences Research Council (EPSRC) for funding the project *libhpc:Intelligent Component-based Development of HPC Applications* stages 1 and 2 (EP/I030239/1, EP/K038788/1) under which this work is being undertaken. DM acknowledges support of the EU project IDIHOM. CC acknowledges support of the British Heart Foundation. We are grateful to the Electrophysiology & Pacing group at Imperial College London for supplying example cardiac electrophysiology data and also thank the Department of Computing at Imperial College London for providing access to their cloud resources, the saga-python team in the RADICAL group at Rutgers, The State University of New Jersey, for supporting our use of their software and the Imperial College London HPC Service for providing cluster access.

References

[1] libhpc: Intelligent Component-based Development of HPC Applications. http://www.imperial.ac.uk/lesc/projects/libhpc. accessed 29^{th} May 2013.

[2] Nektar++ spectral/hp element framework. http://www.nektar.info/, 2013. accessed 1^{st} June 2013.

[3] E. Afgan, D. Baker, N. Coraor, B. Chapman, A. Nekrutenko, and J. Taylor. Galaxy CloudMan: delivering cloud compute clusters. *BMC Bioinformatics*, 11(Suppl 12):S4, 2010.

[4] Alair Engineering. PBS Pro. http://www.pbsworks.com/. accessed 31^{st} May 2013.

[5] Amazon Web Services, Inc. Amazon Elastic Compute Cloud (Amazon EC2), Cloud Computing Servers. http://aws.amazon.com/ec2, 2013. accessed 1^{st} June 2013.

[6] V. R. Basili, D. Cruzes, J. C. Carver, , L. M. Hochstein, J. K. Hollingsworth, M. V. Zelkowitz, and F. Shull. Understanding the high-performance-computing community: A software engineer's perspective. *IEEE Software*, 25(4):29–36, 2008.

[7] J. Cohen, A. S. McGough, J. Darlington, N. Furmento, G. Kong, and A. Mayer. RealityGrid: an integrated approach to middleware through ICENI. *Phil. Trans. R. Soc. A*, 363(1833):1817–1827, 2005.

[8] K. W. et al. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Research*, 2013. First published online May 2, 2013, doi:10.1093/nar/gkt328.

[9] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers. Examining the challenges of scientific workflows. *Computer*, 40(12):24–32, 2007.

[10] T. Goodale, S. Jha, H. Kaiser, T. Kielmann, P. Kleijer, A. Merzky, J. Shalf, and C. Smith. A Simple API for Grid Applications (SAGA), version 1.1. GFD-R-P.90, Open Grid Forum, SAGA-CORE-WG, 2011. Available at http://www.ogf.org/documents/GFD.90.pdf.

[11] J. Cohen, C. Cantwell, D. Moxey, P. Burovskiy, J. Darlington, S. Sherwin. Undertaking Finite Element Analyses on Cloud Platforms for libHPC. Unpublished manuscript, in preparation.

[12] Josh Marshall. jsonrpclib. https://github.com/joshmarshall/jsonrpclib. accessed 29^{th} May 2013.

[13] W. Lee, A. S. McGough, and J. Darlington. Performance evaluation of the gridsam job submission and monitoring system. In *UK e-Science All Hands Meeting*, pages 915–922, Nottingham, UK, September 2005.

[14] M. E. Pierce, R. Singh, Z. Guo, S. Marru, P. Rattadilok, and A. Goyal. Open community development for science gateways with apache rave. In *Proceedings of 2011 ACM Workshop on Gateway Computing Environments*, GCE '11, pages 29–36, New York, NY, USA, 2011. ACM.

[15] R. Ramos-Pollan at al. Bigs: A framework for large-scale image processing and analysis over distributed and heterogeneous computing resources. In *E-Science (e-Science), 2012 IEEE 8th International Conference on*, pages 1–8, Chicago, IL, USA, October 2012.

[16] S. Marru at al. Apache airavata: a framework for distributed applications and computational workflows. In *Proceedings of 2011 ACM Workshop on Gateway Computing Environments*, GCE '11, pages 21–28, New York, NY, USA, 2011. ACM.

[17] S. Shahand at al. A data-centric science gateway for computational neuroscience. In *Proceedings of the 5th International Workshop on Science Gateways*, IWSG 2013, Zurich, Switzerland, 2013. CEUR-WS.

[18] M. Slawinska, J. Slawinski, D. Kurzyniec, and V. Sunderam. Enhancing portability of HPC applications across high-end computing platforms. In *Parallel and Distributed Processing Symposium, 2007. IPDPS 2007. IEEE International*, pages 1–8. IEEE, 2007.

[19] The BioHPC project. BioHPC. http://biohpc.org/. accessed 29^{th} May 2013.

[20] The SAGA Project. SAGA-Python: A Light-Weight Access Layer for Distributed Computing Infrastructure. http://saga-project.github.io/saga-python/, 2013. accessed 29^{th} May 2013.

[21] N. Wilkins-Diehr. Special issue: Science gateways - common community interfaces to grid resources. *Concurrency and Computation: Practice and Experience*, 19(6):743–749, 2007.