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 Sherwin†

*Department of Computing, Imperial College London, South Kensington Campus, London SW7 2AZ, 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 of computational platforms continue to grow, scientific software is becoming ever more complex in order to target these platforms effectively. When using large-scale distributed infrastructure such as clusters and clouds it can be difficult for end-users to make efficient use of these platforms. In the libhpc project we are developing a suite of tools and services to simplify job description and execution on heterogeneous infrastructure. In this paper we present Nekkloud, a web-based software environment that builds on elements of the libhpc framework, for running the Nektar++ high-order finite element code on cluster and cloud platforms. End-users submit their jobs via Nekkloud, which then handles their execution on a chosen computing platform. Nektar++ provides a set of solvers that support scientists across a range of domains, ensuring that Nekkloud has a broad range of use cases. We describe the design and development of Nekkloud, user experience and integration with both local campus infrastructure and remote cloud resources enabling users to make better use of the resources available to them.

I. INTRODUCTION

As computing 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 efficiently on such computing resources is a far from trivial task and usually involves understanding highly technical aspects of the underlying system configuration and hardware. Often, end-users do not have access to HPC resources locally due to financial or administrative reasons, or they may require resources larger than those available to them on-site. Cloud computing environments provide an avenue for on-demand HPC capabilities, but come with their own set of technical hurdles in terms of resource preparation and deployment.

The challenges associated with using HPC resources have been acknowledged previously in various scientific fields, particularly 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 [13] presents the user with a web interface for deploying computational biology simulations on Windows HPC clusters. In statistics, a web application has been developed to deploy R scripts to a set of local HPC resources [9]. A common drawback amongst these solutions is that they do not offer the flexibility to target both cluster and cloud platforms. Furthermore, to the best of the authors’ knowledge, there exists no such environment for finite element applications.

In this paper we present Nekkloud, a web-based software environment to support the use of the Nektar++ [2] high-order finite element framework on both clusters and clouds by scientists and engineers across a range of domains. Nekkloud aims to bundle the support that computer scientists and method developers may provide to scientists, clinicians or engineers wanting to run large parallel Nektar++ computations, into a user-friendly web-based environment that gives control to the end-users and provides a much easier route to perform large-scale scientific simulations on clusters and cloud platforms. Nekkloud uses aspects of the libhpc framework [1], in particular the libhpc deployment service. Libhpc is a framework being developed to simplify the execution of HPC codes in distributed, heterogeneous computing environments.

In this work we have extended the libhpc deployment 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. By exploring two real-world use-cases, we demonstrate how the ability to deploy software to different resources, through a centrally accessible and easy-to-use interface, opens up new opportunities for scientists, clinicians and other non-technical users to leverage Nektar++ in their application areas.

We consider the main contributions of this paper to be:

- The development of a modular and extensible software environment to enable scientists to target large-scale computation on either cluster or cloud infrastructure without the need for a detailed understanding of the underlying hardware and system configuration.
- Integration of the deployment service with existing campus HPC resources along with local and commercial cloud infrastructure.
- A web-frontend module for the exemplar application Nektar++, a high-order finite element code, and demonstration of an improved user experience when running Nektar++ jobs.
In section II we outline the current challenges for non-technical users in using finite element codes on HPC facilities, describe the usage of Nektar++ and further elaborate on the motivation for 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 different components which are used to construct the system. We then discuss the experience for an end-user using Nekkloud in section IV before summarising our work.

II. CHALLENGES & MOTIVATION

A. Current Challenges for FEM Users

To date finite element codes have typically been used by scientists who have sufficient computer science expertise to run the code locally on a desktop system or on homogeneous compute clusters provided by their institution. Consequently, users lacking this expertise are unable to run the code without support from a computer scientist, or someone with specialist computing knowledge. For example, in one of our use-cases, a clinical cardiac electrophysiologist may wish to perform a patient-specific simulation of the electrical activity in a patient’s heart in order to aid diagnosis and plan clinical intervention. The clinician is unlikely to be trained in the use of HPC facilities, finite element methods or performance-optimizing code for a particular target architecture, but must necessarily use such resources in order to obtain results within the prescribed clinical timeframe. Ideally, a black-box system is needed for which the clinicians must provide only domain-specific knowledge. Current mechanisms for using HPC facilities require a high level of technical expertise, precluding direct use by such non-technical users.

Even for technically trained users, modern HPC facilities are becoming increasingly complex and difficult to use if maximum performance is to be attained. The traditional route for performing finite element analysis using HPC facilities is through submission of jobs to queues on local or remote clusters using a batch submission system. If the software is not available on the target platform, the user may need to install binaries or build the software from source code. Once the software is available on the target platform, the user must upload the necessary files to the cluster’s filesystem, write a job submission script, submit the job and wait until the necessary resources are available before the job will execute. A user typically works with only a few clusters (usually local systems and possibly a national facility), each of which may use a different queuing system and be configured differently. This can require changes to parameters in the input files and changes to job submission parameters in the submission script (e.g. filesystem location, site-specific policies on how to use the system) in order to run the same job on each of these clusters, further discouraging the user from diversifying their infrastructure usage and making the most efficient use of the available computing resources. All of the above must almost always be achieved using command-line tools, even for some graphically-driven commercial codes. This presents a barrier to many non-technical users making use of such software.

From the institution perspective, the cost of maintaining clusters of ever-increasing size, suitable for running today’s large-scale finite element simulations, is becoming more prohibitive due to initial hardware costs, data centre costs and ongoing power and maintenance requirements. Cloud computing environments can provide an alternative model for obtaining additional computing power, without the long-term costs associated with running a local cluster, but are currently not prevalent for finite element simulations due to the complexity of their use. To use these platforms, virtual machines running user-customised software images are started on demand and terminated after a job is complete. The complexity of configuring the virtual machine images, installing the necessary software and initialising the jobs presents a further barrier to many users, who do not know (or want to know) how to start cloud resources, configure virtual machines and deploy their code in this fashion.

Finally, in parallel to the increasing capacity of local cluster and cloud computing environments, the underlying hardware architecture is becoming more complex with the prevalence of many-core computing devices such as GPUs and the Intel® MIC architecture, with complex memory and communication hierarchies. Besides the inevitable programming challenges this presents to software developers, it also creates further difficulties for users who need to target their job to the correct hardware types so that it runs with maximum efficiency. Software may also require complex configuration options to target such heterogeneous hardware - decisions which the user may not have the expertise to make.

To address these challenges, we propose the need for a software layer between the end-user of the finite element code and the computing resources on which the code is to be run, to handle the deployment of a user’s jobs to a range of infrastructure resources and ensure the application code is tuned to achieve maximum performance.

We next provide some background about the target application and look at the key aims in the design of Nekkloud.

B. Nektar++

Nektar++ [2] is a powerful open-source application 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. At the core of the software is a library of FEM-related functions, on top of which a range of PDE-specific solvers have been developed. Our aim has been to design a system that enhances the user experience of working with Nektar++ solvers while providing much greater flexibility in the type and size of computing platforms that can be used to run a user’s jobs. Despite the availability of detailed build instructions for Nektar++ and a build system based on CMake [11], the complexity of the software with its many dependencies and the wide range of supported platforms means that problems are likely to be encountered when building the software from source. Nonetheless, building from source allows users to ensure that they gain optimal performance by using the most suitable libraries and compiler optimisations available on a given platform. The range of solvers provided with Nektar++ includes two solvers that will be shown in the example Nekkloud use-cases discussed in section IV – an incompressible Navier-Stokes solver that is used to model fluid flow and a cardiac electrophysiology solver used by clinicians to simulate electrical activity in a patient’s heart.
Nektar++ takes XML input data describing a one-, two- or three-dimensional mesh and a set of problem parameters. For time-dependent problems the selected solver then time-integrates an initial condition on the given mesh, evolving the solution over a prescribed number of time steps and outputs the solution. Optionally, intermediate solution states can be stored during the running of a job, allowing the simulation to be restarted and enabling animated visualisations to be produced. Nektar++ was selected as a key target application for the libhpc project because its internal design consists of a number of modules and associated choices that must be made about how to carry out the solution of a problem depending on its type, size and a range of other parameters [6], [12], [15]. The software makes extensive use of BLAS routines and there are multiple implementations of core FEM operations which can be selected to achieve maximum efficiency on a given hardware architecture. Selecting the correct input parameters and implementation choices is critical to ensure that the user receives optimal performance. Even minor changes in performance of particular routines can make a significant difference to the overall runtime of a user’s job, if it runs for many hours.

C. Aims of Nekkloud

Nekkloud is a web application providing a browser-based user interface for handling job execution and management on clusters and clouds. The user interface is designed specifically to support the Nektar++ high-order finite element analysis software but much of the underlying implementation is designed to be generic so that additional front-end modules may be developed to support the running of other HPC applications from different domains on heterogeneous distributed computing platforms. Nekkloud builds on services from the libhpc framework and is being developed as part of the libhpc project.

Ultimately, ensuring the performance of codes like Nektar++ remains acceptable across a range of different platforms requires substantial domain knowledge and an understanding of the code and hardware. That is not to say that other users cannot make effective use of the software without this support, but where large and long-running jobs are concerned, getting the most out of both their hardware and the code is likely to be a key user requirement.

Many academic users now have access to institutional or departmental HPC facilities and the emergence of public clouds provides a further opportunity to utilize large-scale infrastructure that individual users may otherwise not have access to. Where institutional clusters are concerned, they are likely to provide scientists with the opportunity to access much larger computational resources than are available in their local environment. With the demand for computational power now present in some form across almost all areas of scientific research, end-users without detailed computing knowledge still regularly want to be able to target these institutional HPC resources. However, the overhead required to learn how to prepare their code and work with the clusters can often make it seem easier to stick with local resources and wait much longer for their computations to complete if dedicated computer science support is not available to them.

In developing Nekkloud we have aimed to tackle several of these issues in order to present new opportunities to scientists working with Nektar++ in the areas of engineering and medicine. While the Nektar++ software and a small number of use-cases are the focus at this stage, we have been keen to ensure that the system we present here is built on a more generic platform. This platform is intended to be capable of supporting a range of other software packages given the development of application-specific interfaces or a more general interface designed to support a range of different types of jobs. The main motivations and aims in the design of Nekkloud are as follows:

- Minimise the computer science training needed by scientists to take advantage of HPC facilities for their research.
- Enable seamless access to both campus HPC infrastructure as well as local and commercial 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.

D. e-Infrastructure and Sustainability

As computation becomes ever more important in industry and science, more attention is being paid as to how the increasingly complex software, hardware and human resources can be sustained over time. The term e-Infrastructure has been coined to refer to this complex ecosystem and the roles the various agencies play within it. We define four general groups of individuals involved in operating and using these e-Infrastructures:

- End-users: The domain specialists who use scientific software as part of their work.
- Method developers: The developers with method expertise and domain knowledge who build the software used by scientists and researchers.
- Computer scientists: Individuals with e-Infrastructure knowledge who can build platforms (e.g. Nekkloud) that allow software provided by method developers to be used by end-users on advanced computational infrastructure.
- Facility providers/operators: Organisations who operate large-scale e-infrastructure platforms providing access to both software and hardware resources.

One of the major challenges in managing and using large-scale e-Infrastructure for scientific software is handling the tightly-coupled dependencies between the different groups that make this possible. Our work in the libhpc project and in developing Nekkloud provides a means of decoupling the tight interactions between these groups that are present in traditional approaches to software and hardware use. This provides great improvements in the long-term sustainability of software, allowing each group to work more independently and introducing elements of flexibility into traditional usage.
scenarios. The run-time dependencies end-users would previously have had on computer scientists and facility providers to help them gain access to platforms, select suitable run-time parameters and develop execution scripts are replaced with the Nekkloud interface and the underlying deployment code that hides as much of the complexity of these tasks as possible while maintaining flexibility to use different platforms and job specifications. Facility providers can register new facilities with a Nekkloud deployment and these then become available as target platforms bringing about the type of transparent technology refresh that infrastructure clouds can offer. Method developers can update their software in a manner that is transparent to the end-user or add new applications and methods, with the user getting benefits such improved performance and bug resolution without having to compile new binaries or alter their local environment. Computer scientists can improve mappings to e-Infrastructure by developing more advanced deployment codes or improving efficiency of platform configuration.

The structure of Nekkloud allows less emphasis to be put on the individual relationships between groups of end-users and technical staff and more emphasis on the ability to provide a set of decoupled layers that help to maintain and evolve the system over the long-term. Incorporation of new software and hardware becomes easier, without blocking end-users from continuing with their work and without the need for them to learn a new interface or set of job parameters since these are handled by the mapping code provided by e-Infrastructure experts. Ultimately, we believe that this approach aids the long-term usability of Nekkloud and systems like it, with the advantage of allowing of new machines and methods to be made available to users without the problems often associated with learning and working with these updated tools.

III. ARCHITECTURE & IMPLEMENTATION

A. Nekkloud design

The general design of the system has been chosen to support and address the motivations and aims described above. An early choice was whether to provide users with a local client application that they would install on their systems or whether to provide a web-based client, which was the chosen approach. A local client would simplify direct communication between a user’s client node and the compute platform(s) available to them. However, a web application would support greater flexibility in allowing users to access their jobs from anywhere. We envisage that a web-based client and its associated server-side application would be installed at an institution level to provide access to local and public infrastructure, rather than a single public instance being provided, which may be unable to access private computing resources secured behind a firewall. For smaller-scale users, the server application could be installed and used on a desktop system.

It was also decided that we would provide Nektar++-specific web-forms rather than attempting to build a generic set of web-forms that could support submission of jobs for many different target applications. This led to the system being named Nekkloud and the ability to provide a wider range of job options that are specific to the Nektar++ application. Other HPC applications could easily be supported by creating a suitable set of web-forms.

Since users often wish to run similar jobs multiple times, we opted to allow them to simply upload complete Nektar++ input files, equivalent to that which they would provide in a command line run, rather than providing an interface to specify and build input files through the Nekkloud web interface. Users are generally familiar with specifying the software parameters but less at ease with configuring and making use of parallel computing infrastructures. It is this task that Nekkloud focuses on simplifying. The system has been designed to provide the following job workflow to users:

1) Visit the Nekkloud web application and log in.
2) Create a new job, specifying:
   • the solver
   • input file(s)
   • computing resource to use
   • number of nodes/processors required
3) Submit the job
4) Monitor live job status if desired
5) Download output data on job completion.

After submitting the job, the user is able to log out and return at a later point to collect completed job output data.

B. Nekkloud System Components

Figure 1 shows the architecture of the Nekkloud system which consists of the main Nekkloud application, our existing libhpc deployment service along with its OpenStack private cloud connector service, and a new PBS Cluster Service. These components are described below.

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 the core data model including all user and job information for the system plus information about input and output data files for jobs.

User views: The user views provide a user-facing representation of data in a web-browser. The views may contain dynamic content obtained from the controller or from the database (via the controller).

REST API: The REST API is designed to be accessed by other libhpc services that need to provide information to, or obtain information from, Nekkloud. For example, job status information can be provided by the deployment service which is then updated in the database for use in user views.

Libhpc Deployment Service: The deployment service handles the configuration and execution of jobs on remote resources. Connectors allow the deployment service to work with different 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 [8].

PBS Cluster Service: The PBS cluster service leverages sagger [14] to allow deployment of MPI-based parallel jobs on PBS-based clusters.
OpenStack Service: The OpenStack service is designed to provision infrastructure cloud resources on our local private OpenStack cloud, configure MPI and then submit and run parallel Nekkloud jobs on these resources. Since the service uses OpenStack’s EC2 interface, it is also capable of working with the Amazon EC2 public cloud service.

It was initially intended to decouple the application logic and data storage from the web application itself. However, this would have resulted in an unreasonable amount of data duplication and challenges in ensuring data consistency since the web application would still require its own user database. It was therefore decided to integrate all of this functionality within the web application which then communicates with the external libhpc services to handle job deployment.

The main external service that Nekkloud communicates with is the libhpc deployment service. This service provides an interface that allows the setup, initialisation and execution of new jobs. Connectors allow the deployment service to support different types of underlying hardware platforms such as PBS clusters and private/public clouds.

The main Nekkloud web application is implemented in Python using the Django Web framework. This provides an MVC (Model-view-controller)-style architecture to the web application, decoupling the way data is displayed from the underlying data model and the logic that determines what data is required for a given view. We use the Bootstrap CSS and Javascript framework to build our web pages. The 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 [10].

C. 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 PBS connector was developed specifically to allow the integration of our central 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 [14] library that provides a Python implementation of the Simple API for Grid Applications (SAGA) [7]. Saga-python provides a series of adapters for connecting to and interacting with remote resources or job submission systems. In the case of PBS, local, SSH and GSISSH communication adapters are provided. We use the PBS SSH adapter to allow us to connect remotely to the interactive job submission node for the target cluster, in this case the Imperial College HPC service, and submit and monitor jobs. Saga-python also provides a series of simple remote directory and file management tools via SSH/SFTP. These tools are used by the libhpc PBS connector for managing job directories and input/output files.

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 libhpc deployment service stores this job information, including the platform to be used for job execution. When the service is notified of a new job, it creates a database entry for it and then passes control to the relevant connector for the selected compute platform. In this case control passes to the PBS Cluster Service component 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. The first task of the PBS connector is to create a job directory on the cluster for this purpose. The saga-python file and directory tools are used to create the directory via SSH. Input data files 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 properties of the job. Saga-python generates a PBS job script based on the properties of this Job object.

For our 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 we made a small
modification to allow this. The pre-execution commands are required to initialise some properties of the cluster environment before the MPI job can begin running and the post execution commands are used to compress all output data into an archive file to be sent back to the application server. Saga-python maintains a status polling thread to monitor the status of a job on the underlying PBS cluster. Once the job is complete, output data files are pulled back to the main application server where local post-processing of results can take place.

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 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. In this section we describe how a user wishing to run a Nektar++ computation interacts with the system and how this experience compares with using Nektar++ outside the Nekkloud environment. We then present two use-cases describing the use of Nektar++, via Nekkloud, in the clinical and engineering domains.

The target audience for the user interface is end-users and the aim in designing it has been to ensure that it is uncluttered and straightforward to use. When first visiting Nekkloud, users are shown a login page (see Figure 2). At present, users wishing to sign up are directed to a member of the Nekkloud team in order to discuss their requirements and obtain an account. Once an account has been configured, users are able to log in and create Nektar++ jobs to run through the system.

Users interact with the system via three main views. The jobs view shows all the jobs a user has run, including those that are in progress and those that have completed successfully or failed. The status view allows a user to view the current status of a given job, including live output of a running job’s progress.

This is likely to be of more use to experienced users but non-technical users can pass the information from the status view to a member of the Nekkloud team in the event of a problem. The job creation view is used to set up a new job.

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 3). 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 our 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. A user can then select a specific resource type and the total number of cores that they would like their job to run on. 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 4 shows the resource types that are available on the Imperial College HPC service clusters when this platform is selected.

One or more Nektar++ input files are selected via the Add files... button in the Upload input files panel. The user then
clicks the Start upload button to begin uploading all input files to the Nekkloud application server. Two checkboxes at the end of the job details form provide the ability to Combine fields and Generate movie output. Depending on the job settings, a set of field files may be generated at a given checkpoint interval during a job run and a final set of output field files are generated on completion of the job. When running in parallel, the computation is split across multiple cores and each core generates a separate group of output field files for its individual partition of the computation. In order to work with the completed output data, these individual partitions must be re-combined into single field files. This is done automatically by Nekkloud if the Combine fields option is selected. If checkpoint field files are generated, these can be converted into frames containing a visualisation of the output. These frames can then be combined to form a video animation and generation of this video is enabled when the Generate movie output box is selected. Once file upload is complete and other job details have been entered, the job can be submitted.

Fig. 5. Main panel of Nekkloud job status page showing a job run on the IC HPC cluster.

A running job can be monitored via the job status page which displays the current status of a job and shows a live feed of output data about the job execution process. At present this box displays extensive debug information during job runs but this can be easily altered to display only general information messages and warnings or errors. Job status updates are obtained from the libhpc deployment service which monitors and controls the job execution process. In the example shown in Figure 5, the job has finished running and it is possible to see links to both input and output files at the bottom of the page. When a job is in the process of running, only the input files are shown. When a job completes successfully, the status page updates to show the output files section at the bottom of the page with links for the user to download these output files (see example visualised output in Figure 6).

Fig. 6. Screenshot of the video output from a Nekkloud Incompressible Navier Stokes job.

When running Nektar++ outside of the Nekkloud environment, a user must find suitable resources and then install Nektar++ on these resources. This may involve obtaining the Nektar++ source code and then compiling and installing the software. Despite Nektar++’s CMake-based build system that can find the necessary dependencies or handle the building of them if they are not present, the complexity of the software and the many differences between platforms, their configurations and local software installations can mean that building from source is a complex process. Once the software is installed the user must write a job-specific submission script if running on a cluster and ensure that parameters within this script are compatible with the target platform. The job must then be submitted via the command line. Once a job has finished running, the user is responsible for gathering the output data and then writing scripts, or editing templates, to post-process this output data as required.

A. Use cases

We now describe two use-cases showing the use of the Nekkloud web interface to launch finite element simulations on cluster or cloud infrastructure. Our aim is to highlight the benefits of the web interface and how it enables use of finite element codes on HPC resources by users who are not equipped with specialist HPC knowledge.

The first example is that of the 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 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 locally managed compute cluster. In the event that the cluster is heavily loaded, the clinician might choose to pay to run the job immediately using public cloud infrastructure. After a few minutes the job completes and the electrophysiologist
views the generated video (see Figure 7) of the simulation to provide insight into the next step of the clinical procedure. The job would have taken many hours to run on a desktop system in the clinical environment.

For the second example we consider a simplified industrial problem of assessing the stability of fluid flow past a bluff body. In our case we choose a simple circular cylinder obstruction and wish to perform a parametric study across a range of flow velocities (or equivalently, Reynolds numbers) to understand when the flow becomes unstable and periodic oscillations occur downstream of the cylinder. If the obstacle were an oil pipeline, for example, this could introduce unwanted mechanical forces on the pipe leading to a shorter lifespan. The engineer prepares a sequence of jobs, configured for the different Reynolds numbers, using the Nekkloud interface. The jobs are dispatched to the local cluster and executed when the necessary resources are available. The engineer can use the web interface to check the status of the jobs. Upon completion of the jobs, simulation output data is available for download from the web interface for further analysis by the engineer.

V. CONCLUSIONS AND FUTURE WORK

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 provides 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. It also simplifies the targeting of complex computations and simplifying common post processing tasks otherwise requiring scripts to be written or modified.

Nekkloud plugs in to existing campus cluster infrastructure allowing users to make use of such services through a modified interface but it also supports connectivity to remote PBS-based clusters or to infrastructure cloud platforms allowing the running of more jobs at once or larger jobs than may previously have been possible with local infrastructure.

Further extensions planned for Nekkloud include automatic platform selection based on user-selected metrics such as cost or time, which is a planned addition to the libhpc framework, and the addition of an administrative interface to allow resource providers to register new platforms with the system.

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 (EP/I030239/1) 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 the example cardiac electrophysiology data and would also like to 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