

An Integrated Cloud Platform for Rapid Interface Generation, Job Scheduling, Monitoring, Plotting, and Case Management of Scientific Applications

Wesley Brewer
Fluid Physics International
Houston, USA
wes@fluidphysics.com

Will Scott
Dept. of Computer Science
University of Washington
Seattle, USA
wrs@cs.washington.edu

John Sanford
Dept. of Horticultural Science
Cornell University, NYSAES
Geneva, USA
jcs21@cornell.edu

Abstract—The Scientific Platform for the Cloud (SPC) presents a framework to support the rapid design and deployment of scientific applications (apps) in the cloud. It provides common infrastructure for running typical IXP (Input-eXecute-Plot) style apps, including: a web interface, post-processing and plotting capabilities, job scheduling, real-time monitoring of running jobs, and case manager. In this paper we (1) describe the design of the system architecture, (2) evaluate its applicability to a scientific workload, and (3) present a number of case studies which represent a wide variety of scientific applications including Population Genetics, Geophysics, Turbulence Physics, DNA analysis, and Big Data.

Keywords—*Platform-as-a-Service (PaaS), population genetics, rapid application development (RAD), scientific computing, cloud computing*

I. INTRODUCTION

Scientific programmers very often invest enormous amounts of time and resources developing sophisticated computer programs for specific technical applications. Many of these programs are both brilliant and elegant, and are potentially of interest to a significant number of users beyond the developers. Most of these programs do not have a broad enough potential user base to justify custom development of sophisticated, easily exported, plug-in, user-friendly input/output, turnkey systems. As a consequence, many wonderful programs are used for merely a few years, by only a handful of people (the developers and their extended team), and then fall into disuse. The utilization of such programs could be greatly enhanced if only they were easily accessed, configured, and operated.

Unfortunately, most scientific programs have been developed in a way that is not easily distributed, nor easily installed, nor easily configured. To make matters worse, there is usually a serious learning curve in terms of using a program. Lastly, many people who might like to use a given program are not themselves programmers – they may be, for example, biologists. For a biologist to understand how to use a program and interpret its output requires an extremely user-friendly user interface. All these problems combine to create an enormous barrier to full utilization of scientific

programs. The learning curve is too long and user pain is too great. The cost of getting started is very often too high, and the benefits are often uncertain. While shared infrastructure is a versatile and powerful resource, few scientists have adopted a Software-as-a-Service (SaaS) mindset.

SPC is designed to allow a scientist to easily upload an executable and a sample input file to our shared infrastructure, and receive a usable and transferable interface. SPC is meant to manage the cloud infrastructure, including an interface to create modified input descriptions, job scheduling, plotting of output data, and file management. We take the realistic view that this is not the primary goal for scientists, and that a successful solution needs first to minimize the work required by users.

In the rest of this paper, we will outline our system for packaging, uploading, configuring, and executing technical programs that would otherwise not see existence beyond their own development team. We illustrate the utility of this system with several modern scientific applications, the primary one being Mendel's Accountant – an advanced numerical simulation of the dynamics of mutation accumulation within biological populations that are undergoing natural selection [1].

II. BACKGROUND

A thorough overview of developments of web-based simulations (WBS) and tools has been well documented by Byrne et al. [2]. In this review, the authors emphasize the development of web simulations in light of more recent prominence of technologies, such as Web 2.0 (including cloud computing), service-oriented architectures (SOA), and the Semantic Web. They mention numerous different types of communications protocols such as using WSDL (web-service definition language) or Java remote method invocation (RMI).

Over the past few years, there have been a number of software packages developed to address the need of running scientific applications on a computer cluster. Wu et al. [3] developed a scientific application framework based on

OpenSocial gadgets. Krishnan et al. [4] developed Opal2, a toolkit which can be used to wrap scientific applications and expose them as web services. Opal2 also provides plugin integration with EC2 and Hadoop. Opal2 provides much of the backend infrastructure for running applications, but relies on other software, such as Kepler for pre-processing, and other codes for post-processing.

During the last couple of years, some new architectures and design methodologies have been proposed for cloud-based simulations. Hu et al. [5] compare four different modern methodologies (simulation model portability [SMP], MyExperiment, NanoHub, and RunMyCode), which promote reuse among common components in cloud-based simulation.

The concept of NanoHub, a scientific hub for web-based simulation for nanotechnology, is based on the HUBZero open source software platform, which uses a typical LAMP-stack (in this case, Linux, Apache, MySQL, PHP) approach for the website and content-management system (CMS), while using a Java-based toolkit called Rappture (Rapid Application Infrastructure) to enable legacy scientific applications to run on the web [6].

Di Pierro [7] developed a python-based web framework called web2py. He uses web2py to show a sample scientific computing application which stores DNA strings and searches for similarities. One of the powerful features of web2py is that it uses its data access layer (DAL) in such a way that many different types of database systems can be supported, including both relational and non-relational models. In fact, SPC uses a version of the DAL from web2py called Gluino.

Liu et al. [8] provide a detailed architecture for Cloud-based Simulation (csim), where they define three key cloud services related to simulation in the cloud: Modeling as a Service (MaaS), Execution as a Service (EaaS), and Analysis as a Service (AaaS). They continue to discuss more efficient ways of scheduling parallel and distributed applications (PADS) and then present four PADS job-scheduling algorithms.

Although other methods exist for creating scientific hubs, they typically require much programming, knowledge and time to set up. To the authors knowledge there are still no available frameworks or middleware solutions that are dedicated to supporting scientific applications in such a way that (1) users can easily upload their program to the cloud, (2) have a user-friendly interface automatically generated for them to run, and (3) provide the infrastructure for all common tasks, such as job scheduling, plotting, file/case management, etc. While SPC will continue to improve in supporting a wider range of application types and also in robustness, the authors desire that it continues to remain faithful to these original goals, especially regarding the

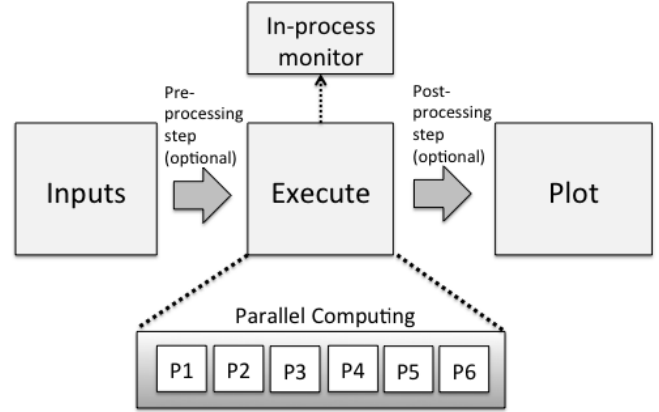


Fig. 1. Many scientific applications fall under an Input-eXecute-Plot (IXP) design pattern.

simplicity of migrating scientific apps to the cloud.

III. SYSTEM ARCHITECTURE

The concept for SPC resulted from identifying the common reusable components in many IXP (Input-eXecute-Plot) style software systems as shown in Fig. 1, such as:

- Interface design
- User authentication
- Job scheduling
- Plotting system
- In-process monitoring
- Management of parallel jobs

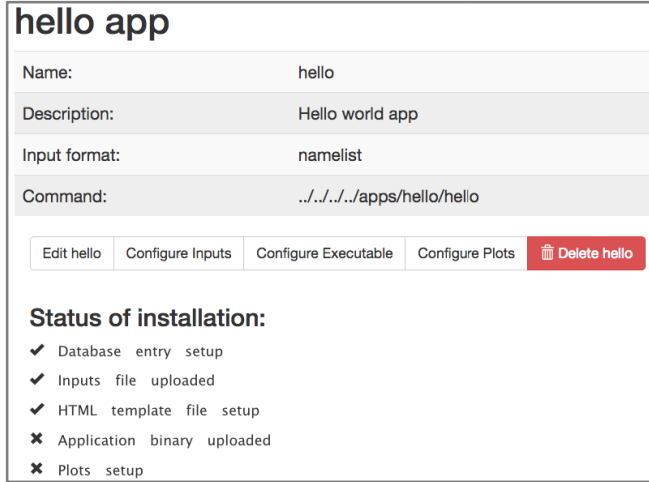
Often, developers use third-party software to handle each of these components. However, the problem with using third-party software was that it made it very difficult to set up the environment on the machine to run the simulation. For example, often OpenPBS (currently rebranded as “Torque”) is implemented as a job scheduler. This one software alone can take quite some time to set up, is non-trivial to manage, and is overkill for managing software running in a single computer node.

By considering a number of similar type software, the following design goals were identified. SPC should:

- Automatically build a web interface
- Manage job execution, scheduling, and monitoring
- Monitor simulation as it is running
- Provide a plotting library interface
- Handle multiple users
- Provide a file/case manager
- Easily be deployed onto Amazon EC2, Google App Engine (GAE), Google Compute Engine (GCE), or RedHat OpenShift.

To meet these goals, the following approach was taken:

- Use a MVT python-based web framework
- Use a simple DB-based scheduler
- Build to support standard scientific application design patterns



hello app	
Name:	hello
Description:	Hello world app
Input format:	namelist
Command:	../../../../apps/hello/hello
Edit hello Configure Inputs Configure Executable Configure Plots Delete hello	
Status of installation:	
✓	Database entry setup
✓	Inputs file uploaded
✓	HTML template file setup
✗	Application binary uploaded
✗	Plots setup

Fig. 2. App configuration page where the admin user can configure the app.

After considering a number of alternative languages, such as Java and Ruby, Python was chosen for three reasons: (1) it has one of the largest scientific computing communities, which includes scientific computing libraries such as SciPy (scipy.org), NumPy (numpy.org), and Matplotlib (matplotlib.org), (2) there are numerous open-source python-based web application frameworks available, and (3) because many of the cloud PaaS providers support Python-based applications (e.g. Google App Engine and Heroku).

IV. SYSTEM DESIGN

In this section we describe each of the main components of the SPC platform.

A. App upload and installation

To fully install a new app requires *four* activities: (1) add the app name, description, input file format, and run command to the main database, (2) upload the default input file and create the HTML input form template, (3) upload and test the binary file, and (4) setup the plots. The initial process may be started by clicking the *Add* button on the *Apps* view, which requires being logged in as the admin user. Once the first step is completed, the new app is added to the database, the user is forwarded to the app configuration page as shown in Fig. 2. From this page, steps 2-4 can be accomplished by clicking the buttons labeled *Configure Inputs*, *Configure Executable*, and *Configure Plots* respectively. This page also provides feedback to the user about the status of installation by using either checkmarks or x-marks to indicate whether each step has been setup properly. It should be mentioned that SPC is able to host any number of apps at the same time.

B. Auto-interface generation.

SPC can be used to automatically generate an HTML

interface given an input deck as shown in Fig. 3. Currently three different types of standard input deck formats are supported: (1) *namelist* input decks which are typically used in Fortran 90 scientific applications, (2) *INI* format which is a standard configuration file typically used in Windows applications, and (3) *xml* format commonly used in Java applications among others.

While the namelist reader/writer had to be custom written for SPC, the INI reader/writer makes use of Python's built-in ConfigParser module. In Fig. 3, we show a portion of the Mendel input deck, and then the HTML template file that SPC automatically generates. Once the HTML template file has been generated, each time the user presses the *Start* button, it populates the template with the default parameters. A previously run case can also be restarted by clicking the *Restart* button which will load the old parameters into the form. Fig. 4 illustrates the create/read/write process for interacting with the HTML template using a parameter named *x*. From the *Jobs* view, the user can restart an old case, which will read the parameters from a previous case and populate the HTML template with those parameters instead.

Finally, there are some applications that will require a customized reader/writer. In these cases, users can write their own Python module with reader/writer methods for reading and writing their unique input deck format.

C. Web Framework

The core of SPC is based on a micro-web server-side framework called Bottle (bottlepy.org), and a front-end framework called BootStrap (getbootstrap.com). This was chosen over a full stack framework to keep the design simple with no external dependencies. Bottle uses a model-view-template (MVT) architecture as shown in Fig. 5, combined with a separate scheduler, and data access layer, which are later discussed. The main purpose of the web framework is to map URL routes to Python methods, but it also provides a simple, yet powerful templating system. While Bottle is not a full stack framework, it is easily extended via many third-party plugins to provide almost any feature that a full stack support, e.g. object-relational mapping (ORM), session management, flash messages, etc.

D. Database

SPC stores all information about users, apps, jobs, and plots in a database. Gluino, a data access layer (DAL) ported from the web2py web framework was implemented so that many different types of databases could be supported, including: SQLite, PostgreSQL, MySQL, Oracle, Google, and many others [9].

The database manages information about currently installed applications, users, and also information about plotting. Fig. 5 shows the general system architecture of the

```
&basic
case_id = test00
mutn_rate = 10.0
frac_fav_mutn = 0.0
reproductive_rate = 2.0
pop_size = 50
num_generations = 100
/
```

HTML
interface

Fig. 3. SPC automatically converts input file to HTML form

SPC web application framework. The model represents the interface to the database, and the views are HTML templates rendered by Bottle's `template()` method.

E. Pre-processing

SPC supports defining pre- and post-processing tasks. Pre-processing might be used for creating a non-standard or customized file that the app may need to for inputs. For example, one app that was tested required all program inputs to be specified on the command line, such as:

```
app.exe -x2 -s2 -n100 -v5 -r10 -k1 -i4
```

In order to use this app with SPC, a standardized `.ini` file was created. Therefore, whenever the user submits a case to run, the pre-processor reads the `.ini` file and maps all the parameters to command line inputs.

In addition to pre-processing, the user can also define post-processing tasks. Post-processing would be typically used to transform data output by the simulation to the correct format needed to plot, or to compute a subset of data from the data generated by the simulation. An example was an app that output its data not in columnar format, but rather as:

```
x1 y1 x2 y2 x3 y3 etc.
```

The post-processing function was used to convert this format into a Python list, which can easily be passed into a

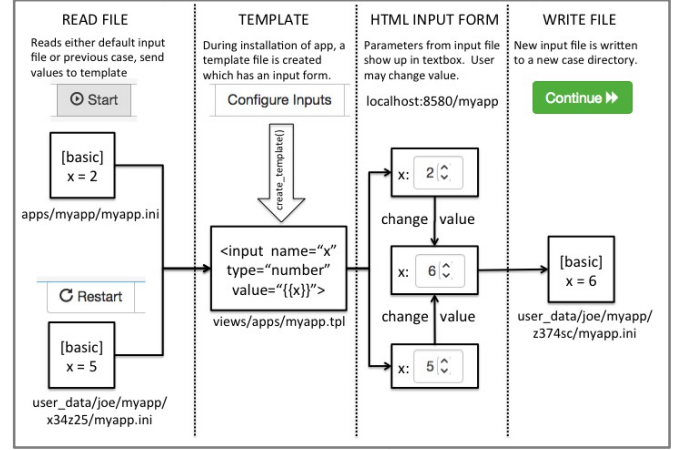


Fig. 4. HTML input form create/read/write process using parameter x .

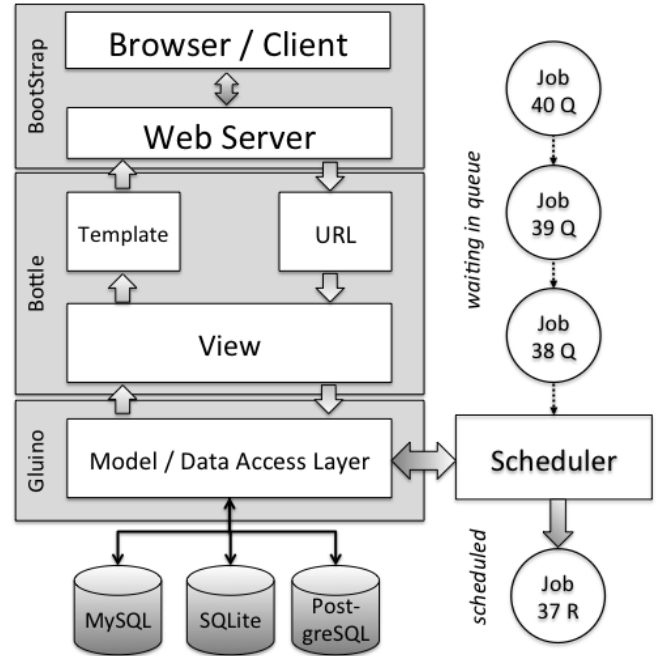


Fig. 5. SPC MVT Architecture similar to Django [10]

plotting library, e.g.:

```
[[x1, y1], [x2, y2], [x3, y3], ... ]
```

F. Job Scheduler

A multiprocessing, priority-based FCFS (first-come first-served) scheduler was developed to manage job submissions from the various apps. The jobs in the queue have *four* possible states: Q for waiting in queue, R for running, C for completed, and X for jobs stopped during run-time. Jobs are submitted to a jobs table in the database, which maintains state information about each job submission.

Users are assigned priority levels, with 0 having the highest priority. The admin user has a priority of 0, while regular users are given a priority of 1, and guest users are given a priority of 2. If several jobs are in the queued state,

the job with the highest priority (or lowest number) is the first to be selected to run.

G. Spawning the app executable

The scheduler has a separate polling thread, which repeatedly polls the database every second and starts executing any job that is in the front of the queue, which is in the queued state. Two possible techniques for spawning new processes have been implemented in SPC: (1) the uni-processor scheduler, and (2) the multi-processor scheduler. First, the simple uni-processor scheduler simply uses the `subprocess` module to start each job by calling the `Pipe()` method. In this case only one job can run at a time. The second method uses the `Process()` method of the `multiprocessing` module so that users can launch more than one job at a time (especially important in the case of multi-user time-sharing of a machine). In the case of multiprocessing, a `BoundedSemaphore` and a `Lock` (mutex) are used to provide synchronization. In the `config.py` file, administrators can specify both (1) which scheduler they wish to use and (2) the maximum number of processors np that SPC is permitted to use. When a job finishes, the state is changed from R to C. Fig. 7 shows a sample output of the job scheduler.

H. Plotting the Data

SPC offers two possibilities for plotting: (1) using a jQuery library called Flot and also (2) using the Matplotlib library to generate static PNG images. Flot is described as “a pure JavaScript plotting library for jQuery, with a focus on simple usage, attractive looks and interactive features” (flotcharts.org). The advantage of using a JavaScript/jQuery library is that all the plotting work is offloaded onto the client, rather than putting the burden on the server, and also the user can dynamically interact with the plot (e.g. zooming). The disadvantage of using Flot is that it has limited support for scientific plots such as contour. On the other hand, Matplotlib was specifically designed for scientific plotting, and supports many different chart types, including color contours. The primary disadvantage of using Matplotlib is that it requires installing about six additional third-party packages, whereas Flot does not require any additional software to be installed.

Regarding the implementation of the plot interface, each plot must be defined in the plot interface dialogue, and then for each plot multiple data sources can be attached to each plot, requiring two tables, *plots* and *data-sources*, with a one-to-many relationship respectively between the two. The plot setup for the Mendel’s Accountant bar-plot shown in Fig. 12 is shown in Fig. 6.

There is also a one-to-many relationship between *apps*

Title:			
Near neutral deleterious effects			
Type of plot:			
<input checked="" type="checkbox"/> flot/line <input type="checkbox"/> flot/categories <input type="checkbox"/> matplotlib/line <input type="checkbox"/> matplotlib/bar			
xaxis label:			
Fitness effect			
yaxis label:			
Frequency			
Options (JSON):			
legend: { position: "nw"}, axis: { axisLabel: "Fitness effect"}, yaxis: { axisLabel: "Frequency" }			

Filename	Columns	Line range	Data definition (JSON)
<cid>.000.hap	1:2	3:103	{ label: "Theoretical", data: d1, color: "rgb(200,0,0)" }
<cid>.000.hap	1:4	3:103	{ label: "Dominant", bars: { show: true, barWidth: 0.0001 }, data: d2, color: "rgb(0,200,0)" }
<cid>.000.hap	1:5	3:103	{ label: "Recessive", bars: { show: true, barWidth: 0.0001 }, data: d3, color: "rgb(0,0,200)" }

Fig. 6. (a) Add plot form and (b) data source interface.

and *plots*, so that the user can define multiple plots for each app. In the plots interface, two types of information are stored as JSON strings: (1) data definition—defines information such as which data will be used, the name of the data to use in the legend, the type of plot (e.g. line-plot or bar-plot), and (2) options—defines options for things like axis labels, etc. Both of these parameters are well defined in the Flot documentation (see flotcharts.org). By storing these parameters as JSON strings, the plot configuration is not limited by the database schema. JavaScript helper functions are used to help automate creation of the JSON strings.

I. Case Management

Each simulation run is assigned a universal unique identifier (UUID) using Python’s built-in `uuid` module. The files and output generated from each run are stored in their own folder in the relative path `user_data/username/appname/caseid` (e.g. `user_data/joe/mendel/c13dxd`).

J. Monitoring the Simulation

Once the simulation has been launched, SPC automatically redirects to the *Case* view. The *Case* view contains a jQuery AJAX call which repeatedly calls a method called `tail`, which retrieves the last 40 lines of the output file every second (e.g. see Figs. 10a, 11, and 12). We are currently working on using a web socket approach for monitoring, which will allow the server to push data to the client thereby reducing client/server communication costs.

★ jid	app	cid	state	np	date/time submitted	labels
★ 716	mendel	c2c552	Q	1	Sat Sep 5 22:51:04 2015	v2.5.1
★ 715	mendel	af04e3	R	1	Sat Sep 5 22:51:00 2015	v2.5.1
★ 714	mendel	i2b242	X	1	Sat Sep 5 22:50:25 2015	v2.5.1
★ 713	mendel	sa8abf	C	1	Sat Sep 5 22:48:49 2015	v2.5.1

Fig. 7. An example of the built-in job scheduler showing jobs in the queue, running and completed jobs, and also the ability to *star* or *share* cases.

K. Worker Management

Integration with the Docker container system is included in SPC in an effort to make deployment as simple as possible. Docker is a platform and API for application deployment that is supported by most major cloud providers including Google, Microsoft, and Amazon (www.docker.com).

When the application finds itself in an environment with a working Docker client, it provides an interface for deploying new workers on which scheduled jobs may run.

Additional interface is shown as part of the input screen, allowing the user to choose how many new worker contexts should be used. After use, workers are automatically stopped, making the deployment process simple enough for non-technical users.

L. Shared Cases

Another feature SPC offers is a way for users to share their results with other users. Since users have access to only their own cases, if they want to share results with others, they can write a comment in the *Jobs* view and click the share icon, after which anyone will be able to see their case, comments, outputs, and even be able to run the same case in their own file space.

M. User and Account Management

Users can register for an account. Passwords are hashed before storage in the database using a 256-bit Secure Hash Algorithm (SHA-256). Two special user account are reserved for *admin* user and *guest* user. The *admin* user allows special features such as installing and configuring apps, but also user account management. The *guest* user is a limited account, which only allows running installed applications, but does not allow app or plot configuration.

V. EXAMPLE APPLICATIONS

SPC is exemplified using a number of different scientific software applications, listed in order of complexity from simple demos to quite complex numerical simulation software:

- 1) *DNA Analyzer*: analyzes a given string of DNA
- 2) *Burger*: a simple finite-difference solution to the inviscid Burger's equation

- 3) *Mendel's Accountant*: a forward-time population genetics simulation program.
- 4) *Terra*: a three-dimensional finite element code for modeling the dynamics of planetary mantles
- 5) *Parspectra*: a parallel spectral method which solves the Navier-Stokes equations in three dimensions
- 6) *Doop*: a simple application for demonstrating the use of Hadoop/MapReduce.

A. DNA Analyzer

DNA Analyzer is a very simple Python program (53 lines of code) that is distributed as an example with SPC. Given an user-defined string of DNA, it will compute:

- Reverse complement
- Percentage content of GC nucleotides
- Dinucleotide frequency distribution
- Codon frequency distribution

The inputs form, and output bar plot are shown in Fig. 8.

B. Solving Burger's Equation

Burger's equation is a simple example of a set of nonlinear partial differential equation called reactive diffusion equations [15]. This equation is typically used in gas dynamics (e.g. to model shock waves) or to model traffic flow. The equation is given as:

$$\frac{\partial u}{\partial x} + u \frac{\partial u}{\partial x} = 0 \quad (1)$$

where u is the speed of the traveling wave in the x -direction. This equation can be solved using the method of finite differences where each derivative is approximated to the first order using a simple finite-difference approximation as follows:

$$\frac{\partial u}{\partial x} \approx \frac{x_{i+1} - x_i}{2} \quad (2)$$

In this program, the user must enter a number of parameters, such as the Courant number, the number of panels to discretize the domain, the length of the domain, and the number of iterations to run. This program would typically be executed in a UNIX terminal window, making it only accessible to a limited number of UNIX/Linux users around the world. By making the program available on the cloud, it instantly becomes accessible to virtually anyone on the planet.

Fortunately, with SPC it only takes a few minutes to transform this console-based application into a web-based simulation. To create the app in SPC, the user must do the following steps:

- 1) The program must read an input file with the parameters. If the input file is already in a standard format (such as namelist.input format, or .ini format) nothing additional must be done. If it is in a non-standard format, the user must either (a) modify the

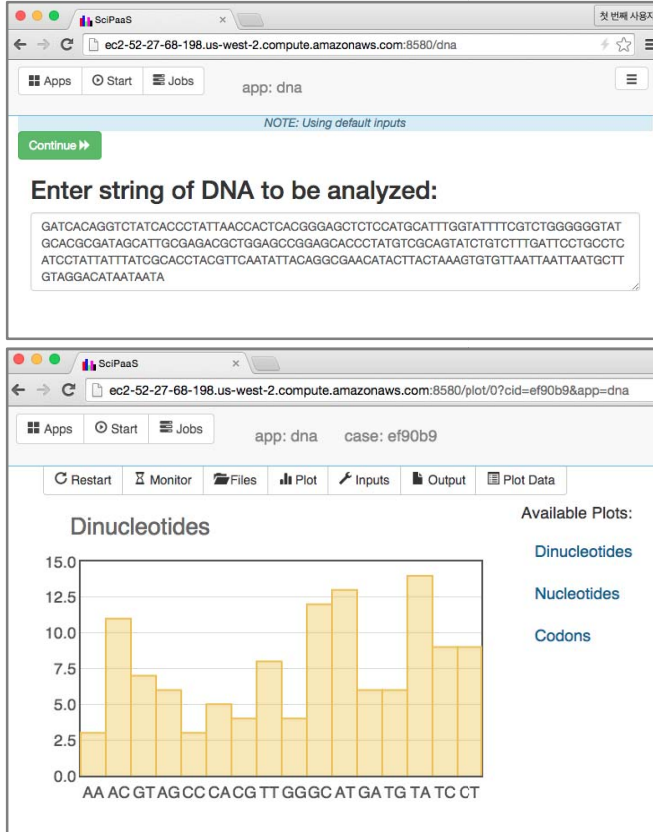


Fig. 8. DNA Analyzer (a) input, and (b) plot of dinucleotide frequencies.

code so that it uses a standard input format, or (b) write a custom reader in Python to read their specific input into SPC.

- 2) Upload a sample user input file with default inputs.
- 3) Upload a binary executable compiled for the specific operating system on which SPC is running.
- 4) Define a Plot. This is simply a matter of clicking which file will be plotted, what columns in the file, and the plot type (e.g. scatter plot, bar chart, etc.).

SPC automatically generates the HTML based on the input deck as shown in Fig. 9a. The next step is to define any plots that should be post-processed. In this case, the plot shown in Fig. 9b represents the shock wave, resulting from solving Burger's equation.

C. Mendel's Accountant

Mendel's Accountant is a forward-time population genetics simulation program, which models genetic change over time. The software is part of a growing trend of many geneticists turning to computer simulation as a promising means to better understand population genetics [11]. Mendel's Accountant is more complicated in two aspects: the simulation engine, and the client interface. The simulation engine is more complicated because (1) it has more than 60 parameter inputs, (2) it is parallelized using the MPI (Message-Passing Interface), and (3) it generates

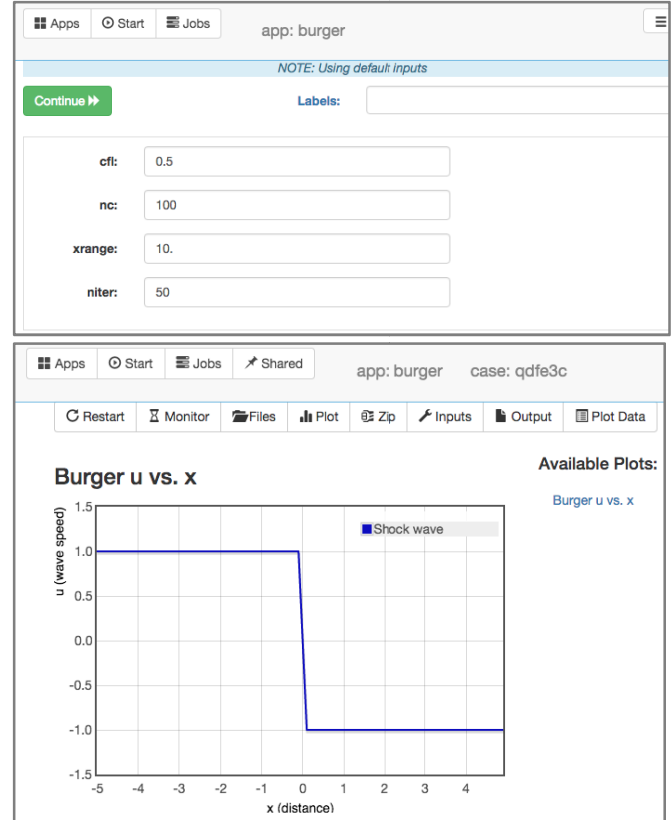


Fig. 9. Burger's (a) inputs and (b) output plot showing traveling shock wave

many output files full of statistical data which need to be plotted. Currently, we have defined nine plots within SPC.

Pre-processing. Even though Mendel's Accountant is quite a complex simulation, since it already supports the standard `namelist.input` input deck, it can be uploaded and running in the cloud in a matter of minutes, as shown in Fig. 3. In this case, after the initial uploading into SPC, we added some further customizations, by adding a number of JavaScript feedback features, so that when one option is selected, they may trigger changes in other values.

Processing. Fig. 10a shows the console output of Mendel's Accountant that is auto-updated via AJAX. In this case, we have also added a progress bar to show a percentage update of the simulation progress.

Post-processing. Once the run has finished and the data needs to be analyzed, clicking the plot button will give a list of pre-defined plots to plot, or can let the user define new plots. Data from multiple sources can be attached to each plot. Fig. 10b shows a bar-plot of the distribution of near-neutral deleterious effects.

The source code was published to the `sourceforge.net` website in 2007. In the intervening eight years it has received 5,000 downloads, but the authors report TODO support emails, and only four follow-on publications not associated with the original development team. One of the motivations for creating SPC was to increase the potential of scientists to adopt software

packages such as Mendel's Accountant by (1) making it more accessible via the cloud, (2) having an intuitive user interface, and (3) lowering the entry barrier by removing any requirements to install or provide a sufficient computer to host the software.

To gain more understanding about the ability of SPC to allow virtually anyone to run a complex scientific computing simulation, we conducted an experiment with a group of 42 volunteers around the world recruited via Facebook.com. Collection of data was gathered in two ways: (1) analyzing the web access logs, and (2) a Google Docs survey form which asked some basic information about their terminal degrees and how well they understood the simulation. 50% of the volunteers had an undergraduate degree as their terminal degree, 19% had a Master's terminal degree, and 19% had a Doctoral terminal degree. In addition to giving them the URL of the EC2, and login credentials, we only gave them one instruction on what to do: *"simulate genetic change over time by running Mendel's Accountant using the default parameters, and then create a plot of near neutral deleterious effects"* (Fig. 6). Out of the 42 visitors, 17 were able to start the simulation, and 16 of them were able to make a plot (however, most only plotted the default plot – Average mutations/individual), representing a success rate of 40%. Given a Likert scale from 1 to 5 about what they understood (1 = *I didn't understand anything about the simulation*, 5 = *I understood the simulation well*), the average result was 2.56. Moreover, a number of very useful comments about how to improve the interface were reported through the survey.

D. Terra

We performed another test in which someone outside our group migrated his scientific software package to the cloud via SPC. In this case, we used Terra, a three-dimensional finite element code for modeling the dynamics of planetary mantles developed by Baumgardner [12]. Baumgardner was interested in using SPC as a means to make his code easily accessible to a wider audience of scientists. Terra represented a more challenging case for SPC due to the complexity of the software. Fig. 11 shows the in-process monitoring of Terra, which is auto-updated via repeated AJAX calls.

Pre-processing. In order to get Terra working as he preferred with SPC, he chose to convert his code to use a standard *namelist.input* format. Also, since SPC used a convention-over-configuration approach for file naming, which requires the main output file to be named *terra.out*, and Terra used a different convention of using an output naming convention such as *out100.00*, we decided to implement a preprocessing step within SPC, requiring an additional five lines of code.

Post-processing. Moreover, since a secondary post-processing code is generally run to output the plots for Terra, Baumgardner decided to modify his code to have his

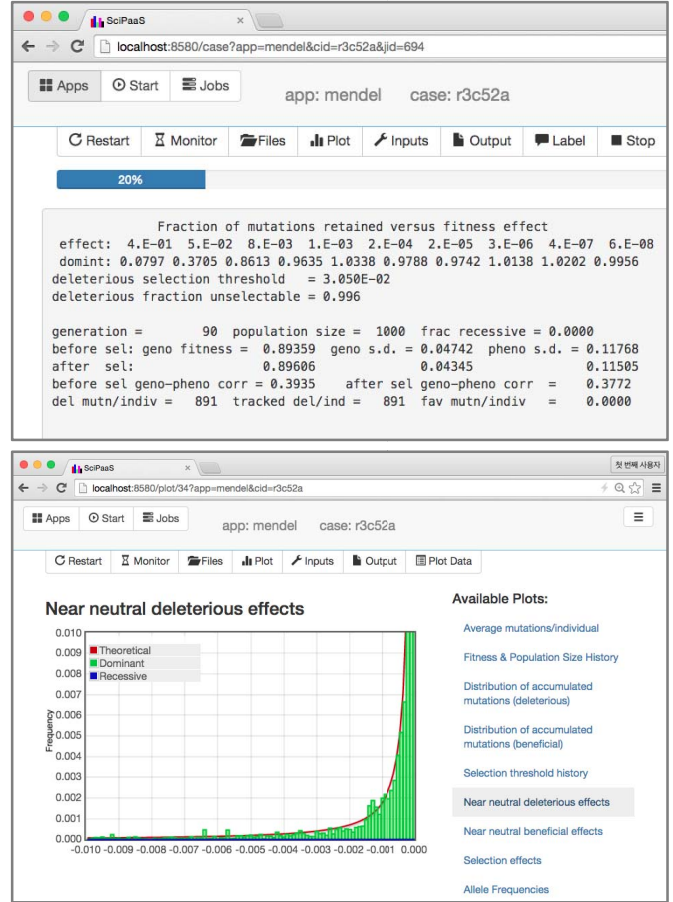


Fig. 10. Mendel's Accountant (a) monitoring output during run, and (b) bar plot of Mendel's Accountant showing near neutral deleterious effects

code directly output graphic images which can be plotted from within SPC.

E. Parspectra

Parspectra is a parallel pseudo-spectral solver for the three dimensional Navier-Stokes equations [13]. It is primarily used in simulating turbulence, and developing new models to predict turbulence in Direct Numerical Simulations (DNS). Parspectra is written in Fortran and is parallelized using the Message Passing Interface (MPI). Fig. 12 shows outputs of Parspectra output while it is running. After running, the energy spectrum can be plotted.

F. Doop

Doop is a simple Hadoop/MapReduce example application that was designed to demonstrate that SPC can also be used for Big Data applications. In order to handle Big Data applications, SPC needed to be modified so that users could upload their data to the server. This option was implemented in the *user account* menu.

Also, Hadoop needed to be installed on the server, and the NameNode and DataNode daemons needed to be up and running. A Bash script was written to handle these

administrative tasks. The actual mapper and reducer were implemented in Python using the Hadoop Streaming API. Then another script was written to handle the general Hadoop workflow: (1) copy data files to be analyzed to HDFS, (2) start the jobs, (3) copy the results back to the local file system. Once the app was setup on SPC, any user can easily run Hadoop-based applications via the web interface.

VI. SETUP AND DEPLOYMENT

For the results for this paper, testing was performed on an Amazon EC2 machine. In the AWS page in SPC, the user may store their Amazon instances, as shown in Fig. 13. Clicking on status provides three functions: *start*, *stop*, and *status*. This feature is implemented via the `boto` package. It will also give the user the public DNS name of the machine. In order to utilize this approach, users must first setup the various machines they might want to use in the future, which would include installing SPC on each machine. Then, the user can start up the big machine (e.g. an `r3.8xlarge` instance which has 32 vCPUs and 244GB of RAM), use it for however long is needed, and then download the case files to an inexpensive `t1.micro` instance, and then shutdown the big, expensive machine.

SPC provides a *Zip* button in the *Case* view, which allows users to compress any case for download either to their home machine, or to another SPC instance.

In the case of MPI-based applications, if the number of processors (parameter `np`) is set to more than one processor in `config.py`, just before the user executes a new run, an option will show up next to the *Execute* button which allows the user to choose how many processors to use (the maximum value being the value of `np` in `config.py`), as:



If the user selects a value greater than one (say, e.g. 2), instead of executing the binary directly, SPC will automatically execute the MPI execute command as:

```
mpiexec -np 2 myapp.exe
```

In this work we have mainly focused on running simulations on a single-node machine. In some more complex numerical simulation systems, e.g. weather forecasting, one may want to provision an entire cluster in the cloud. Currently, this kind of cluster provisioning must be done outside the context of SPC. SPC may be readily extended to start MPI jobs using a number of nodes in the cloud. However, this feature has not yet been implemented.

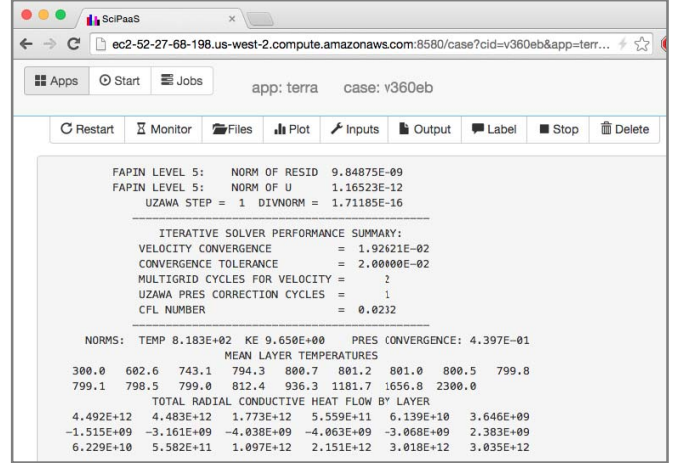


Fig. 11. In-process monitoring of Terra mantle convection code

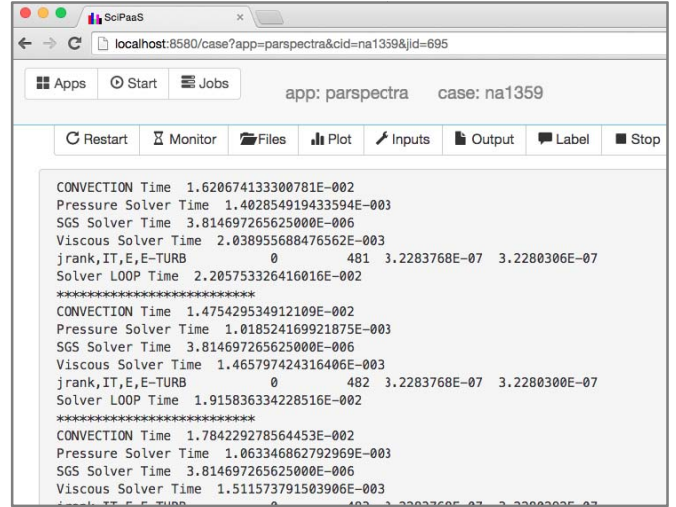


Fig. 12. In-process monitoring of Parspectra

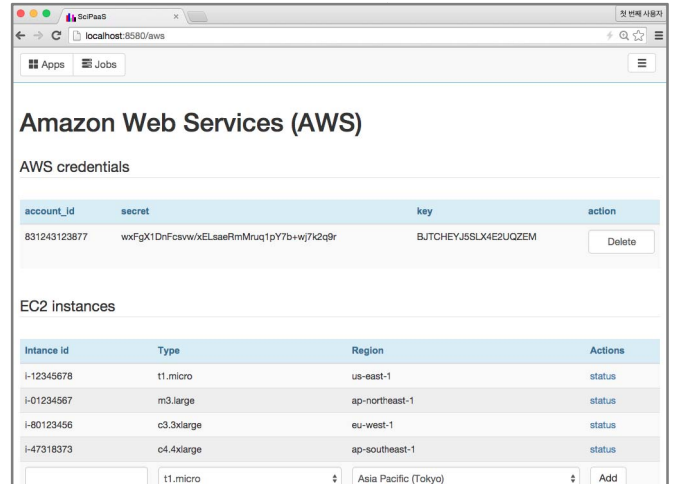


Fig. 13. AWS configuration page showing how users can enter their AWS credentials and instances

VII. CONCLUSIONS AND FUTURE WORK

A middleware execution platform called SPC was described and demonstrated with several different scientific software applications. The platform was developed to meet a niche need that is not currently available, that is, a PaaS that is specific to hosting scientific simulation programs that require specific needs, such as job scheduling, plotting, in-process monitoring, case and file management.

By providing an automatically generated easy-to-use interface, and an easy way to upload and plug-in their applications to the platform, SPC allows scientists to rapidly deploy their applications to the cloud. An unintended benefit of this type of platform solution to web-based simulations is that it also *encourages* developers of scientific simulations towards using standard protocols where appropriate, such as using standardized input formats.

SPC has been tested with a number of scientific software packages, including finite element analysis codes, Hadoop/MapReduce codes for working with big data, in addition to a number of bioinformatics codes. The software is available in the Open Source domain online at <https://bitbucket.org/whbrewer/spc>.

Future work includes making SPC more robust by providing support for more complex pre- and post-processing tasks, but also providing additional support for more complex plotting capabilities, such as contour plots. Also, plans include implementing a workflow management system whereby it could handle more complex software simulation packages. Finally, we are working on a command-line installer that will automatically download and install scientific apps from a central repository in a completely automated way. Such apps would be those submitted to a central repository by other SPC users.

ACKNOWLEDGMENT

W.H.B. would like to thank *FMS Foundation* and *GCC Jackson* for the funding graciously provided for this project. Also, thanks to Tichomir Tenev of VMWare for reviewing the manuscript, and also to John Baumgardner and Shanti Bhushan for testing SPC with their codes, and finally to Alice Houston for proofing the manuscript.

REFERENCES

- [1] J. Sanford, J. Baumgardner, W. Brewer, P. Gibson, and W. ReMine, "Mendel's Accountant: A biologically realistic forward-time population genetics program," *Scalable Computing: Practice and Experience*, vol. 8, no. 2, July 2007, pp. 147-165.
- [2] J. Byrne, C. Heavey, and P. Byrne, "A review of Web-based simulation and supporting tools," *Simulation modeling practice and theory*, vol. 18, no. 3, 2010, pp. 253-276, doi:10.1016/j.simpat.2009.09.013.
- [3] W. Wu, T. Uram, M. Wilde, M. Herald, and M. Papka, "A Web 2.0-Based Scientific Application Framework", *Proc. IEEE International Conference on Web Services*, IEEE Press, July 2010, pp. 642-643, doi:10.1109/ICWS.2010.107.
- [4] S. Krishnan, L. Clementi, J. Ren, P. Papadopoulos, and W. Li, "Design and Evaluation of Opal2: A Toolkit for Scientific Software as a Service", *Proc. 2009 IEEE Congress on Services-I*, July 2009, pp. 709-716, doi: 10.1109/SERVICES-I.2009.52.
- [5] C. Hu, C. Xu, G. Fan, H. Li, and D. Song, "A Simulation Model Design Method for Cloud-Based Simulation Environment," *Advances in Mechanical Engineering*, vol. 5, no. 932684, Aug. 2013, doi:10.1155/2013/932684.
- [6] M. McLennan and R. Kennell, "HUBzero: A Platform for Dissemination and Collaboration in Computational Science and Engineering," *Computing in Science and Engineering*, vol. 12, no. 2, March 2010, pp. 48-52, doi:10.1109/MCSE.2010.41.
- [7] M. Di Pierro, "web2py for Scientific Applications," *Computing in Science and Engineering*, vol. 13, no. 2, March 2011, pp. 64-69, doi:10.1109/MCSE.2010.97.
- [8] X. Liu, X. Qiu, B. Chen, and K. Huang, "Cloud-based Simulation: the State-of-the-art Computer Simulation Paradigm", *Proc. 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation*, July 2012, pp. 71-74, doi:10.1109/PADS.2012.11.
- [9] M. Di Pierro, "web2py Complete Reference Manual (6th edition)", Retrieved from <http://www.web2py.com/books/default/chapter/29/06/the-database-abstraction-layer>.
- [10] Z. Hong, "The Django Web Application Framework", Retrieved from <http://www.slideshare.net/fishwarter/the-django-web-application-framework-2-1221388>.
- [11] J. Sanford and C. Nelson, "The Next Step in Understanding Population Dynamics: Comprehensive Numerical Simulation, *Studies in Population Genetics*", Dr. M. Carmen Fusté (Ed.), ISBN: 978-953-51-0588-6, InTech, Aug. 2012, pp. 117-136, doi:10.5772/34047.
- [12] J. Baumgardner, "Three-dimensional treatment of convective flow in the Earth's mantle," *Journal of Statistical Physics*, vol. 39, no. 5, June 1985, pp. 501-511, doi:10.1007/BF01008348.
- [13] S. Bhushan, and D. K. Walters. "Development of Parallel Pseudo-Spectral Solver Using Influence Matrix Method and Application to Boundary Layer Transition," *Eng. Applications of Computational Fluid Mechanics*, vol. 8, no. 1, Nov. 2014, pp. 158-177, doi:10.1080/19942060.2014.11015505.