SciPaaS: A Scientific Execution Platform for the Cloud

Wesley H. Brewer[[1]](#footnote-1) , John C. Sanford[[2]](#footnote-2) , Will Scott

*Abstract*— SciPaaS is a prototype development of an execution platform/middleware designed to make it easy for scientists to rapidly deploy their scientific applications (apps) to the cloud. It provides all the necessary 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 even a file/case manager. In this paper, first the system architecture is described and then is demonstrated for a two scientific applications: (1) a simple finite-difference solver of the inviscid Burger’s equation, and (2) *Mendel’s Accountant*—a forward-time population genetics simulation model. The implications of the prototype are discussed in terms of ease-of-use and deployment options, especially in cloud environments.

*Keywords*—web-based simulation (WBS), cloud computing, platform-as-a-service (PaaS), rapid application development (RAD), population genetics

# INTRODUCTION

S

cientific 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 just 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 just a few years, by just 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 cobbled together 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 how to interpret the 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.

We outline here a generic system for the easy packaging, uploading, configuration, and utilization of any technical program that would otherwise not find application beyond the development team. We illustrate the utility of this system using the technical program called *Mendel’s Accountant* – an advanced numerical simulation of the dynamics of mutation accumulation within biological populations that are undergoing natural selection [1].

With the promise of on-demand computing access, cloud computing has become an invaluable resource for scientists in general. The problem is that most scientists have little knowledge about the cloud, and much less about how to build a Software-as-as-Service (SaaS). The concept of SciPaaS is that a scientist could easily create a zip archive of their code containing just the binary executable and a sample input file, upload it to the cloud, and SciPaaS would manage all the cloud infrastructure for them, including the input interface, job scheduling, plotting, etc. This allows the scientists to focus on developing software to solve the problem at hand, without having to worry about the added overhead of developing the infrastructure for the execution and interface environment.

# Background

Some of the earliest efforts to do web-based simulations trace back to the 28th Winter Simulation Conference held in 1996 [2]-[4].

A thorough overview of developments of web-based simulations (WBS) and tools has been well documented by Byrne et al. [5]. 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).

More recently, there has been a number of software packages developed over the past few years to address the need of being able to run scientific applications on a computer cluster. Wu et al. [6] developed a scientific application framework based on OpenSocial gadgets. Krishnan et al. [7] developed Opal2, a toolkit basically 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 years, there some new architectures and design methodologies have been proposed for cloud-based simulations. Hu et al. [8] 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 [9].

Di Pierro [10] developed a python-based web framework called web2py. He uses web2py to show a sample scientific computing application in which stores DNA strings and searches for similarities. One of the powerful features of web2py is that it uses in data access layer (DAL) such that many different types of database systems can be supported, including both relational and non-relational models. In fact, SciPaaS uses the DAL from web2py.

Liu et al. [11] 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). Then they discuss more about more efficient ways of scheduling parallel and distributed applications (PADS) and then present four PADS job scheduling algorithms.

In this paper, we propose similar goals as with HUBZero and Rappture. However, the goal is to have no external dependencies other than Python, and as such can be quickly deployed on any platform. Moreover, an input/output/plot interface is automatically created for each application. This also builds upon the cloud simulation architecture proposed by Liu et al. [11], but we focus more on the front-end development rather than the overall architecture. More specifically, we focus on developing both a framework and a platform for rapidly deploying an existing or legacy simulations to the cloud.

Although other methods exist for creating scientific hubs, they typically require much programming, knowledge and time to setup. To the authors knowledge there are still no available framework or middleware solutions that are dedicated to supporting scientific applications in such a way that: (1) a user 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.

# System Architecture

The concept for SciPaaS basically came out of developing a Scientific Software-as-a-Service (SaaS) interface for a simulation engine called Mendel’s Accountant [1]. Mendel’s Accountant is a biologically-realistic, forward-time population genetics simulator which can be used for studying mutation accumulation dynamics in genomes. During the development of this application a number of key ingredients required for general scientific applications were identified, which we have termed an IXP (Input-eXecute-Plot) pattern as shown in Fig. 1. The concept for SciPaaS resulted from identifying the common reusable components in many IXP style software systems, such as:

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

During the development of Mendel’s Accountant, third-party software was used to handle each of these components. The problem with using third-party software was that it made it very difficult to setup the environment machine to run the simulation. For example, one of the necessary components, a job scheduler, was implemented using OpenPBS (currently rebranded as “Torque”). This one software alone can take quite some time to setup and also is non-trivial to manage.

Moreover, by considering a number of similar type software, some design goals were identified as follows. SciPaaS 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 design goals, the following approach was used:

* Use a MVT python-based web framework
* Use a simple DB-based scheduler
* Deploy to free PaaS providers such as Heroku, OpenShift, Google App Engine (GAE), etc.
* Build to support standard scientific application design patterns

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). Furthermore, in the design of SciPaaS, a convention-over-configuration philosophy was implemented, such that a number of the files (i.e. the binary executable, the default input deck filename, the name of the view template, etc.) all use the name appname.

Inputs

Execute

Plot

P1

P2

P3

P4

Parallel Processing

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

Monitor

# Discussion

SciPaaS features are demonstrated for a typical scientific application called Mendel’s Account, an advanced numerical simulation program for modeling genetic changes over time. The simulation engine of Mendel’s Accountant was developed in Fortran 90 because of its ability to do numerical computation very efficiently.

## Upload app to cloud

A zipfile containing a default input file and binary of the application is uploaded to SciPaaS. The upload process unzips the file to the appropriate locations, reads the default input deck, and then creates an HTML template file views folder named the same as the application. The next section explains how the interface is generated from the input deck.

## Auto-interface generation.

SciPaaS can be used to automatically generate an HTML interface given an input deck as shown in Fig. 2. Currently three different types of standard input deck formats are supported: (1) *namelist* input decks which are typically used in Fortran 90 scientific applications (e.g. NCAR’s Weather Research and Forecasting [WRF] software), and (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 SciPaaS, the INI reader/writer makes use of Python’s built-in ConfigParser module. In Fig. 2, we show a portion of the Mendel input deck, and then the HTML template file that SciPaaS automatically generates.

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

## Web Framework

The core of SciPaaS is based on a micro-web framework called Bottle (bottlepy.org). This was chosen over a full stack framework to keep the design simple with no external dependencies. Bottle uses a Django-like MVT (model-view-template) architecture as shown in Fig. 3 [12]. 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 3rd party plugins to provide almost any feature that a full stack support, e.g. object-relational mapping (ORM), session management, flash messages, etc.

## Database

SciPaaS stores all information about users, apps, jobs, and plots in a database. A data access layer (DAL) from the web2py python web framework (dal.py) was implemented so that many different types of databases could be supported, including: SQLite, PostgreSQL, MySQL, Oracle, MSSQL, Google, Firebird, DB2, Informix, Ingres, Cubrid, Sybase, Teradata, SAPDB, MongoDB, and IMAP [13].

The database manages information about currently installed applications, users, and also information about plotting. Fig. 3 shows the general system architecture of the SciPaaS web application framework, which uses an Model-View-Template (MVT) architecture. Basically the model represents the interface to the database, and the views are essentially HTML templates rendered by Bottle’s template method in a simple way. For example, to render a plot, we can use a simple command such as:

return template('plot', params)

Here, plot refers to the plot.tpl file in the views folder and params contain a Python dictionary of some parameters about the app, the case to plot, and the user.

MENDEL input deck

HTML interface

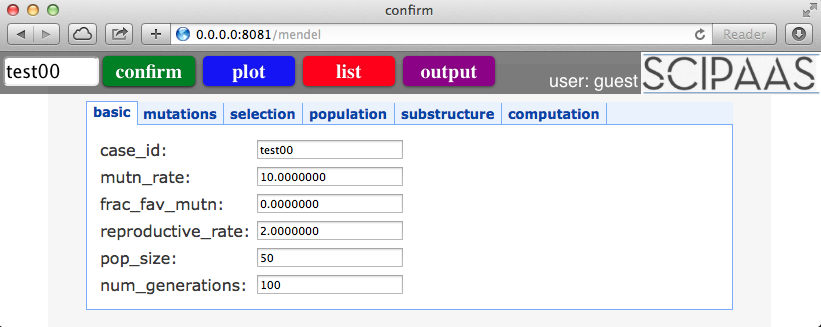


Fig. 2 SciPaaS automatically converts namelist format input file to HTML form.

## Executing the Simulation Engine

There are several possible ways to spawn the simulation engine from within Python. One option is to use the subprocess module, which supports either a call method or a pipe. Another method for spawning the engine is simply to use the system call from the os module. The important point is that the job must be launched in the background so that SciPaaS can continue to handle requests. The way this is handled is by spawning a new thread for every new job that is submitted. The standard output stdout of the simulation is redirected to the file *appname*.out (e.g. mendel.out in the current example).

DataBase

Model

View

Template

URL dispatcher

Brower

Fig. 3 MVT Architecture

## Case Management

Each simulation run is assigned a universal unique identifier (UUID) using Python’s built-in uuid module. It is important to have a way to identify each simulation run via SciPaaS for several reasons:

1. Identifiers are needed as a primary key in the jobs table.
2. Virtually every function related to cases and case data are referenced using this identifier.
3. The files and output generated from each run is stored under the relative path user\_data/appname/caseid (e.g. wes/mendel/c13dxg).

## Monitoring the Simulation

Once the simulation has been launched, SciPaaS automatically redirects to the monitor view. The monitor view is essentially a jQuery AJAX call which repeatedly calls a method called tail, which retrieves the last 40 lines of the output file every second.

## Plotting the Data

SciPaas offers two possibilities for plotting: (1) using a jQuery library called Flot, and also (2) using the Matplotlib library to generate static PNG images. The advantage of using a JavaScript or 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 the JavaScript library is it supports only basic line, points, bar, and category charts, and was not specifically designed for scientific plotting (e.g. could not use it for a color contour plot). However, 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.

At any time during the simulation or after running, the user can plot data from the simulation. When the user clicks the plot button, the first defined plot will be displayed with options to show other possible plots. The user can then choose to plot one of the available plots, or can click the Edit button to define a new custom plot.

Flot is described as “a pure JavaScript plotting library for jQuery, with a focus on simple usage, attractive looks and interactive features” (flotcharts.org). In order to prepare the data for plotting, a plot class is defined within SciPaaS. This class contains a method called get\_data, which extracts the necessary data from the files, and returns it in JSON format. An example of plotting fitness vs. history in Mendel’s Accountant is shown in Fig. 4.

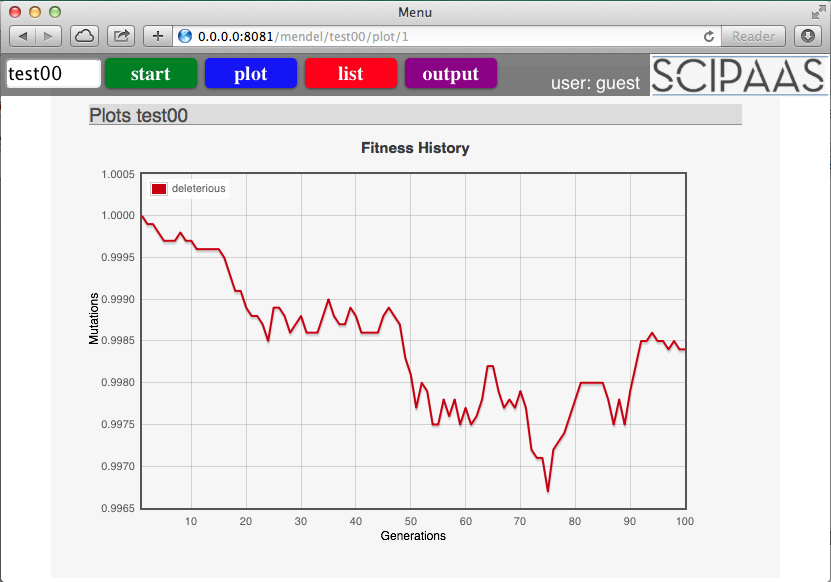


Fig. 4 Example of jQuery/JavaScript plot for Mendel.

## Job Scheduler

A simple FCFS (first-come first-served) scheduler was developed to manage job submissions from the various apps. Jobs are submitted to a jobs table in the database, which maintains state information about each job submission. The scheduler has a separate polling thread, which repeatedly polls the database every five seconds and starts executing any job that is in the front of the queue which is in the queued state. The jobs in the queue have three possible states: Q for waiting in queue and R for running or C for completed. Each job is run in a separate execution thread. When the execution thread finishes, the job is immediately deleted from the jobs table. The following figure shows a sample output of the job scheduler:

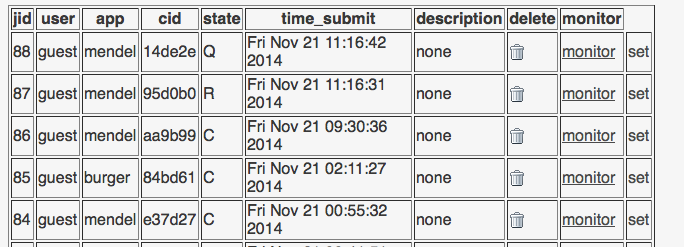


Fig 5 An example of the built-in job scheduler showing jobs in the queue, running and completed jobs

## The Wall

The final feature of SciPaaS is a results “wall”, whereby users can share their results with one another. 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 “post to wall”, after which anyone will be able to see their case, comments, outputs, and even be able to run the same case in their own filespace.

## Worker Management

Integration with the Docker container system is included in SciPaaS in an effort to make deployment of 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 to deploying new workers for scheduled jobs to run on. 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.

# Example Applications

SciPaaS is exemplified using two different scientific software applications as follows:

1. **Burger** – a simple finite-difference solution to the inviscid Burger’s equation
2. **Mendel’s Accountant** – a forward-time population genetics simulation program.

## Solving Burger’s Equation

Burger’s equation is a simple example of a set of nonlinear partial differential equation called reactive diffusion equations [14]. This equation is typically used in gas dynamics (e.g. to model shock waves) or to model traffic flow. The equation is given as:

(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:

(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 SciPaaS it only takes a few minutes to transform this console-based application into a web-based simulation. To create the app in SciPaaS, 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 code so that it uses a standard input format, or (b) write a custom reader in Python to read their specific input into SciPaaS.
2. Upload a Zip-compressed version of the program, with a sample user input file.
3. 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.).

SciPaaS automatically generates the HTML based on the input deck as such:

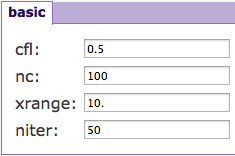


Fig 6 HTML parameter input for Burger's equation

The next step is to define any plots that should be post-processed.

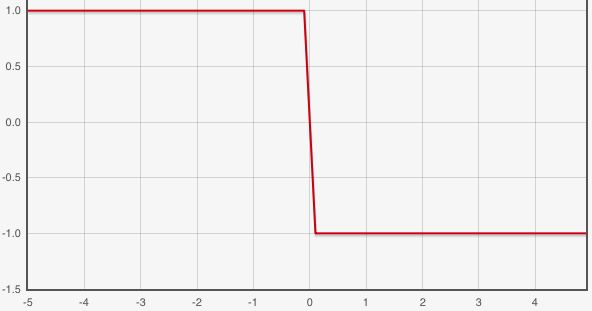


Fig 7 Burger’s equation output plot showing traveling shock wave

In this case, the plot represents the shock wave.

## B. Mendel’s Accountant

Mendel’s Accountant represents a much more complex example, a 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 [15]. 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 to the simulation, (2) it is parallelized using the MPI (Message-Passing Interface), (3) generates many output files full of statistical data which needs to be plotted.

*Pre-processing*. Even though Mendel’s Accountant is a quite complex simulation, since it already supports the standard namelist.input input deck, it can be uploaded and running in the cloud in just a matter of minutes. The following figure shows the full input parameter screen:

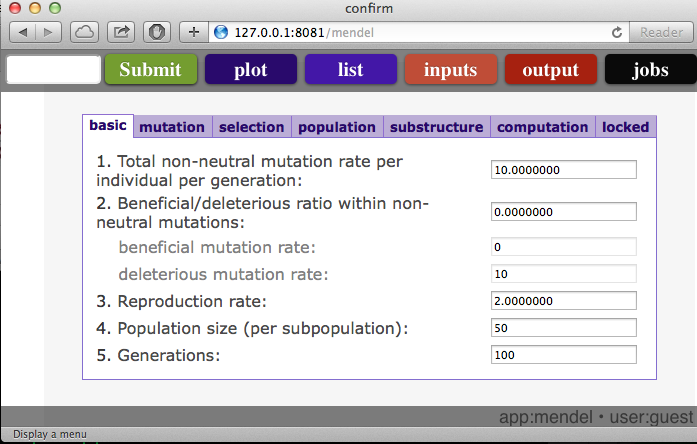


Fig 8 Input parameter screen for Mendel’s Accountant

*Processing*. The following figure shows the console output of Mendel’s Accountant that is auto-updated via AJAX:

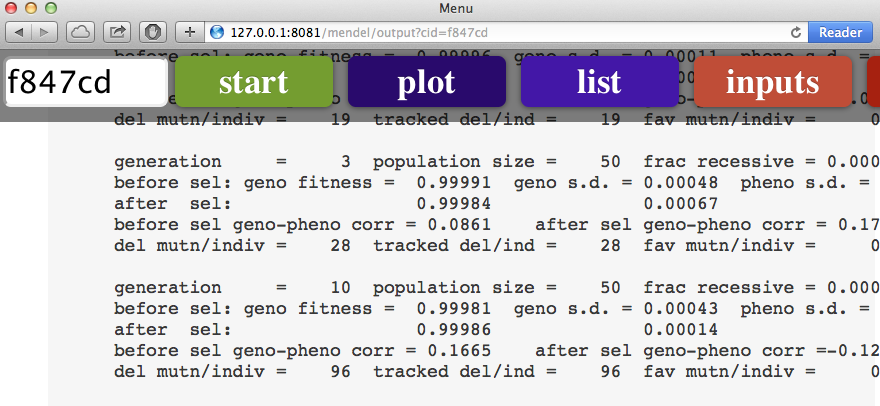
****

Fig 9 In-process monitoring of Mendel’s Accountant simulation

*Post-processing*. Once the run has finished and the data needs to be analyzed, clicking the plot button will given a list of pre-defined plots to plot, or can let the user define new plots:

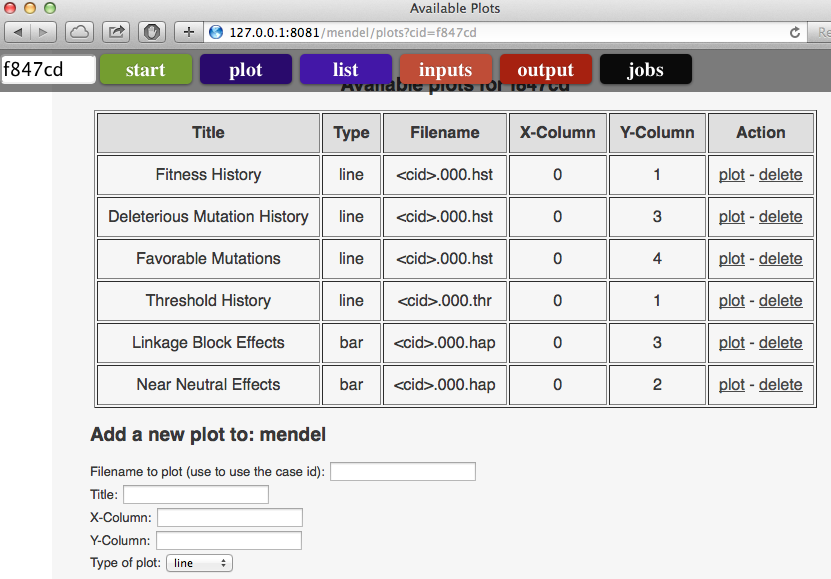
****

Fig 10 Plot configuration screen

The following figure show one of the plots generated using the Flot jQuery library. As an example, clicking the plot button will generate the following fitness history and mutation accumulation figures respectively.

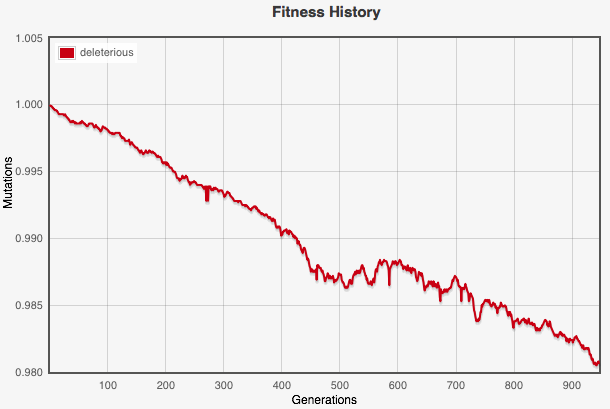
****

Fig 11 Sample output plot of Mendel’s Accountant showing fitness history

# Deployment

SciPaaS can be deployed in a number of ways:

1. Run SciPaaS on your personal computer to host a number of locally installed applications. This is a helpful option for development and unit or functional testing.
2. Run SciPaaS on a traditional Linux server.
3. Run SciPaaS on an Infrastructure-as-a-Service machine (e.g. EC2 instance on Amazon Web Services or Google Compute Engine, etc.). This would be the best option in that it could support binaries compiled in any language.
4. Run SciPaaS on top of an existing PaaS, (e.g. Google App Engine [GAE], AppScale, etc.). The limitation here would be that it would only be able to support app binaries that the PaaS supports. For example, running on GAE would require apps to be written in Java, Python, Go, or PHP.

The authors have tested SciPaaS both on each of these options except the last option. We have recently developed a cloud test platform using a LEAP cloud stack (Linux, Eucalyptus, AppScale, and Python) in order to test running SciPaaS on top of a generic Platform-as-a-Service solution such as AppScale.

# Conclusion

A middleware execution platform called SciPaaS was described and demonstrated with both a simple Burger’s equation solver, as well as Mendel’s Accountant, a more complex forward-time population genetics simulator. The software will soon be released in the Open Source domain online at https://github.com/whbrewer.

By providing an automatically generated easy-to-use interface, and an easy way to upload and plugin their application to the platform, SciPaaS 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.

What SciPaaS currently lacks is the ability to run as a production-level server, scaling up to a web scale user base. In order for SciPaaS to scale, it needs to use a high-performance, multi-threaded web server such as NGINX. Furthermore, SSL will need to be implemented as well for securing the transmission of data between the client and server. One possible solution to many of these issues is to rebuild the SciPaaS platform on top of a full stack web framework such as web2py—a Python-based web framework specifically designed with rapid development of scientific applications in mind.

In addition, much future work remains, such as:

* Deployment to Google App Engine
* Implementing a generalized workflow to be able to handle custom-defined and more generalized workflows
* Interfacing with other web services, such as the Opal2 toolkit
* Supporting parallel execution infrastructure, such as interfacing with MPI and Hadoop.

Acknowledgment

The authors want to acknowledge appreciation for the support graciously provided by both *FMS Foundation* and *GCC Jackson* organizations for this project. Also, many thanks to Tichomir Tenev from VMWare for reviewing the manuscript and providing helpful feedback.

References

1. Sanford, J., Baumgardner, J., Brewer, W., Gibson, P., & ReMine, W. (2007). Mendel's Accountant: A biologically realistic forward-time population genetics program. *Scalable Computing: Practice and Experience*, *8*(2).
2. Fishwick, P.A. Web-Based Simulation: Some Personal Observations. *Proceedings of the 1996 Winter Simulation Conference*, Coronado, CA (1996): 772-779.
3. Nair, R. and Miller, J.A. A Java-Based Query Driven Simulation Environment. *Proceedings of the 1996 Winter Simulation Conference,* Coronado, CA 786-793.
4. Buss, A. and Stork, K. Discrete Event Simulation and World-Wide Web Using Java. *Proceedings of the 1996 Winter Simulation Conference*, Coronado, CA (1996): 772-779.
5. Byrne, James, Cathal Heavey, and Peter J. Byrne. "A review of Web-based simulation and supporting tools." *Simulation modeling practice and theory* 18.3 (2010): 253-276.
6. Wu, W., Uram, T., Wilde, M., Herald, M., and Papka, M. “A Web 2.0-Based Scientific Application Framework”, IEEE 2010.
7. Krishnan, S., Clementi, L., Ren, J., Papadopoulos, P., and Li, W. Design and Evaluation of Opal2: A Tookit for Scientific Software as a Service. 2010
8. Hu, C., Xu, C., Fan, G., Li, H., Song, D. “A Simulation Model Design Method for Cloud-Based Simulation Environment. Advances in Mechanical Engineering, <http://dx.doi.org/10.1155/2013/932684> (2013).
9. McLennan, M.; Kennell, R., "HUBzero: A Platform for Dissemination and Collaboration in Computational Science and Engineering," *Computing in Science and Engineering*, **12**(2):48-52 (2010).
10. Di Pierro, Massimo, “web2py for Scientific Applications.” *Computing in Science & Engineering* 13.2 (2011): 64-69.
11. Liu, X., Qiu, X., Chen, B., Huang, K. “Cloud-based Simulation: the State-of-the-art Computer Simulation Paradigm”, 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation (2012).
12. Hong, Zhixiong, “The Django Web Application Framework” http://www.slideshare.net/fishwarter/the-django-web-application-framework-2-1221391 (2009).
13. Di Pierro, Massimo, web2py Complete Reference Manual (6th edition), http://www.web2py.com/books/default/chapter/29/06/the-database-abstraction-layer.
14. Debnath, Lokenath. "Nonlinear partial differential equations for scientists and engineers." Birkhäuser (2012).
15. John C. Sanford and Chase W. Nelson (2012). 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, DOI: 10.5772/34047.

1. W. H. Brewer is with Fluid Physics International, Houston, TX 77266 USA (phone: +1-713-893-8700; e-mail: wes@fluidphysics.com) [↑](#footnote-ref-1)
2. J. C. Sanford is with the School of Integrative Plant Science, NYSAES, Cornell University (e-mail: jcs21@cornell.edu)

   W. R. Scott is with the Department of Computer Science & Engineering, University of Washington (e-mail: wrs@cs.washington.edu) [↑](#footnote-ref-2)