**SciPaaS: a Scientific Execution Platform for the Cloud**

**SciPaaS: a Python-based execution platform for rapidly deploying scientific applications to the cloud**

**Sixpence: a scientific integrated execution platform environment for cloud applications**

Wesley H. Brewer1 and John C. Sanford2

**ABSTRACT**

SciPaaS is 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 typical scientific application, Mendel’s Accountant—a forward-time population genetics simulation model.

**1. INTRODUCTION**

With the promise of on-demand computing access, cloud computing has become an invaluable resource for scientists in general. The only 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 an interface.

There have been a number of software packages written over the past few years to address the need of being able to run scientific applications on a computer cluster. Wu et al. (2010) developed a scientific application framework based on OpenSocial gadgets. Unfortunately, the code is not open source, so cannot be freely downloaded and used. Krishnan et al. (2010) 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. However, the author has not been able to find any packages to simply upload an app and start to run on the cloud.

**2. 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 (Sanford et al, 2007). 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 was identified, which we have termed an IXP (Input-eXecute-Plot) pattern as shown in Figure 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 not-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, (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.

**3. DISCUSSION**

Now SciPaaS features are demonstrated for a typical scientific application called Mendel’s Account, an advanced numerical simulation program for modeling genetic changes over time (Sanford et al., 2007). 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. SciPaaS currently can handle two different types of standard input deck formats: (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.

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 Figure 2, we show a portion of the Mendel input deck, and then the HTML template file that SciPaaS automatically generates.

Inputs

Execute

Plot

P1

P2

P3

P4

Parallel Processing

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

Monitor

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](http://www.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. 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.

**Database**. SciPaaS stores all information about users, apps, jobs, and plots in a SQL database. SQLite3 is used as the primary database system to manage information about currently installed applications, manage users, and also manage information about plotting. Figure 3 shows the general system architecture of the SciPaaS web application framework, which uses an Model-View-Template (MVT) architecture. Basically the model reqpresents 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

MENDEL input deck

HTML interface

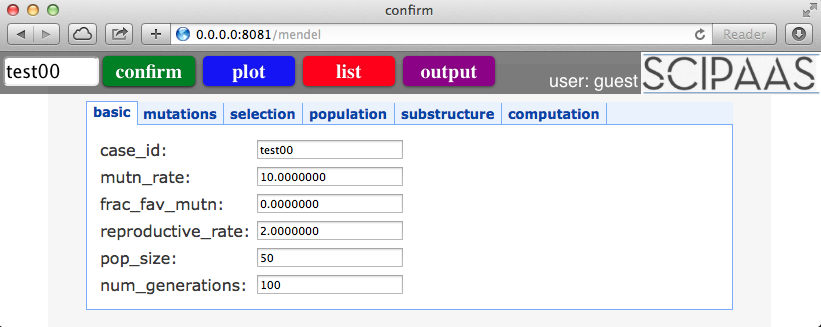


Figure 2. SciPaaS automatically converts namelist format input file to HTML form.

DataBase

Model

View

Template

URL dispatcher

Brower

Figure 3. MVT Architecture

parameters about the app, the case to plot, and the user.

**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 to 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).

**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. Figure 5 shows an example of the output monitor, in which generation 50 is enveloped with a red box to highlight the movement of the output stream up the screen.

**Plotting the data**. At any time during the simulation or after running, the user can plot data from the simulation. When the user clicks the plot button, a list of defined plots is returned. The user can then choose to plot one of the available plots, or can define a new custom plot.

The plotting system is handled using a jQuery library called flot ([www.flotcharts.org](http://www.flotcharts.org)), which they describe as “a pure JavaScript plotting library for jQuery, with a focus on simple usage, attractive looks and interactive features”. 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

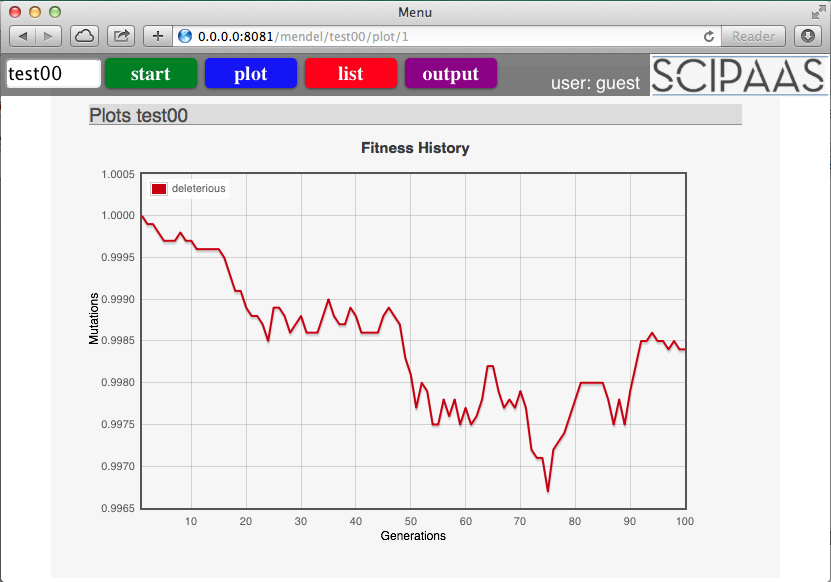


Figure 4. Example of jQuery/JavaScript plot for Mendel.

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 Figure 4.

**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 two possible states Q for waiting in queue and R for running. Each job is run in a separate execution thread. When the execution thread finishes, the job is immediately deleted from the jobs table.

**4. CONCLUSIONS**

A middleware execution platform called SciPaaS was described and demonstrated with Mendel’s Accountant, a forward-time population genetics simulator. Future work includes:

* Easy deployment to a PaaS provider
* Implementing a generalized workflow to be able to handle custom-defined workflows
* Interface to manage virtual machines, such as an administration view for starting and stopping additional EC2 machines.
* Interfacing with other web services, such as the Opal2 toolkit
* Supporting parallel execution infrastructure, such as interfacing with MPI and Hadoop.

**5. ACKNOWLEDGMENTS**

The authors want to acknowledge appreciation for the support graciously provided by the FMS Foundation for this project.

**6. REFERENCES**

1. Wu, W., Uram, T., Wilde, M., Herald, M., and Papka, M. “A Web 2.0-Based Scientific Application Framework”, IEEE 2010.
2. Krishnan, S., Clementi, L., Ren, J., Papadopoulos, P., and Li, W. Design and Evalution of Opal2: A Tookit for Scientific Software as a Service. 2010
3. 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).
4. “The Django Web Application Framework” by Zhixiong Hong 2009.3.26. http://www.slideshare.net/fishwarter/the-django-web-application-framework-2-1221391.

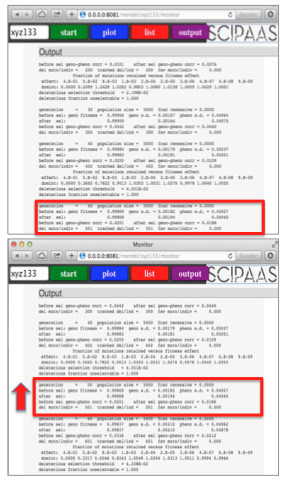


Figure 5. Monitor view showing AJAX-based auto-updating of simulation output.

Review:

This approach presented in the paper seems well implemented and I can see the need for it. However, I feel the author is underselling the work in the following ways (this also has a negative impact on my perception of the paper's contribution):  
1- there is no real lit review reducing the scientific weight of the paper.   
2- the use case, although meaningful, is somewhat arbitrary, and there is no way for the reader to determine if SciPaaS does its job as defined in the intro well or not. What are the performance metrics that demonstrate that SciPaaS is good tool? This needs to be made clearer and integrated into the discussion.  
3- This is a 4 page submission, but I think padding this paper out a bit with more details, a solid evaluation, and better thought out case study, and a decent lit review would make it a better (and more citable) paper. I find myself stuck on the question: what the author's motivates are for this paper. If it's a quick stake in the ground then the paper certainly does that, but more than that is unlikely...   
  
Minors:  
- fig 5 is not readable.  
  
On the whole I think this paper has promise, and for a workshop paper it's ok as a WiP, but I struggle to see exactly what the paper contributes to the literature in its current form.

|  |  |
| --- | --- |
| Relevance to workshop | Good |
| Technical Soundness | Good |
| Presentation | Good |
| Reviewer Expertise | Knowledgeable |
| Overall Rating | Neutral |
| Confidence | Confident |

The list of features in sec. 2 suggests an ambitious project, however I am missing key elements, namely what's in my input zip file? surely there are requirements for what SciPaaS can deal with? Sec. 3 delves into technical details using one example which is based on a Fortran application, but it does not directly answer the question.  
  
Other details regarding the developers' level of awareness of the framework are not very clear, for instance under "database", we find "to render a plot, we can use simple commands such as..." --> what does what exactly? do I as a developer need to write Python code, and if so to what extent exactly?  
  
minor: figs sizes are off -- 3 is oversized, 5 is unreadable

|  |  |
| --- | --- |
| Relevance to workshop | Some |
| Technical Soundness | Medium |
| Presentation | Medium |
| Reviewer Expertise | Knowledgeable |
| Overall Rating | Neutral |
| Confidence | Medium |