Scalable cloud platform middleware for integrating bioinformatics algorithms

Wes Brewer
Computational Solutions LLC
wes@computational.io

Abstract—

Background: In this paper we propose a cloud platform middleware for running bioinformatics software in the cloud called SPC—the Scientific Platform for the Cloud.

Results: We demonstrate how such a platform can be used for teaching, research, and benchmarking algorithms. We also show that SPC is useful for doing side-by-side comparisons of software with similar functions. We demonstrate SPC on various cloud service providers such as Amazon Web Services Elastic Compute (EC2) Platform, Google App Engine PaaS, but also using Docker Containers orchestrated by Kubernetes running on Google Compute Engine.

Conclusions: We find that the platform is a very convenient and useful way for scientists to easily migrate their apps to the cloud for utilization by the broader scientific community. This platform facilitates collaboration with others, and also works well for teaching Bioinformatics in the classroom**.**

Keywords — Cloud Computing, Middleware, Bioinformatics, Integration of Algorithms, Simulation

I. Background

Why the cloud? Over the past ten years cloud computing has created a paradigm shift in computing practice, but not without good reason. There are enormous benefits to be gained by taking advantage of this technology. We list five main benefits we have experienced in using the cloud here [1]:

1. **Massive storage capabilities** – these days, scientists are working with datasets that are on the order of Petabytes [2]. This is almost impossible to do on conventional desktop machines.
2. **Massive parallelism** – using technologies such as the Message Passing Interface (MPI) and MapReduce
3. **Easy collaboration** with other scientists – by having all the data in the cloud it is easy to share data without having to rely on e-mail.
4. **Software installation & maintenance** – users do not have to worry about installing or maintaining software.
5. **Virtual machines on demand** – we can easily start up larger compute machines, as we need them.

In this paper we explore the question: can we build a cloud platform (SPC) that can both leverage the power of the cloud while at the same time lowering the entry barrier to cloud adoption for developers and users of Bioinformatics codes? We will first give a literature review of previous work. Then, we introduce SPC, its architecture, features, and typical usage patterns. Then, we describe a number of bioinformatics applications (apps) we have ported to run on the platform, which include apps for sequence analysis, sequence alignment, and more complex apps such as for simulating population genetics. We show how the platform is useful for teaching, doing research, and benchmarking algorithms.

Dudley and Atel [3] give a good overview of using cloud computing for Bioinformatics research. They discuss the issue of reproducibility of scientific data in the cloud and propose a concept, which they coin as "whole-system snapshot exchange" (WSSE) as a means to replicate an entire computational system in order to exchange with other researchers.

Schatz et al. [4] give a brief overview of technologies that are available in the cloud for analyzing high-throughput NGS data, such as using a "Map-shuffle-scan" technique based on Hadoop/MapReduce framework.

With the advent of next-generation sequencing (NGS), the amount of genomic data is being harvested at "population scale" resulting in about 40 petabytes every year [2]. For a small research lab, the only plausible way to deal with such huge amounts of data is in the cloud. Therefore, there has been a move especially in the field of computational genetics to migrate to the cloud.

Recently there have been a number of cloud-based tools and platforms that have been developed to handle sequenced genomic data. SequenceServer is a modern Ruby-based interface for running BLAST searches through the web [5], which uses the Sinatra web framework. The Broad Institute of MIT & Harvard's Genome Data Analysis Center (GDAC) maintains a website, firebrowse.org, for easily interacting with TCGA data. They also have an API that provides users to programmatically access the data using various programming languages, such as Python and R [6].

Web-TCGA is a web-based portal for more easily working with The Cancer Genome Atlas (TCGA) dataset, and is built using R in conjunction with the Shiny web application framework [7].

While there are many cloud-based tools for doing a variety of computational genetics analysis in the cloud, such as sequence alignment, etc. there has not been a platform for hosting various types of bioinformatics software, such as complex population genetics simulators, in the cloud. This was the main motivation of this research.

II. Implementation

This system was built on SPC by Brewer et al., which is more thoroughly discussed in [8]. SPC provides a number of useful features as listed here:

- *Auto-interface generation* so that the simulators can be rapidly migrated to the cloud. Basically all we need to do is upload a sample input file, and SPC will automatically generate the web interface. SPC supports the following standard file formats: INI, XML, JSON, YAML, Namelist.input.
- *Plotting capability* using open source plotting libraries, such as flot (flotcharts.org).
- A built-in *job scheduler* for scheduling jobs. This manages the jobs, which are running on the computer, so that the computer does not get overloaded with too many running simulations.
- Live *monitoring* of running simulations.
- Ability to do *cloud bursting* using larger Amazon EC2 instances through SPC's AWS management interface
- Support for *parallelism* using MPICH, MapReduce, or OpenMP.
- Has a *case and file management system*, which automatically stores each run into separate directories. Users can label runs with various tags, easily view any data files, zip and download files, diff files all from the web interface.
- Built-in *Docker container* management from within SPC, so that a user can manage images, start containers, and even launch jobs on a local or remote containers.
- Ability to run on almost any platform, including *AWS EC2*, Docker *containers* and *Google App Engine* as shown in Figure 1.
- Ability to easily *share* cases and data with other users on the same machine.
- Built-in *package management system* for easily installing apps from the web, using a command such as: `spc install http://foo.com/app.zip`

The architecture for SPC is based on the commonly known Model-View-Template (MVT) architecture and is based on a customized Python-based framework that is a combination of the Bottle web framework and Web2py's data access layer (DAL) as shown in Figure 2. The reason for using a DAL was so that it could readily be ported to

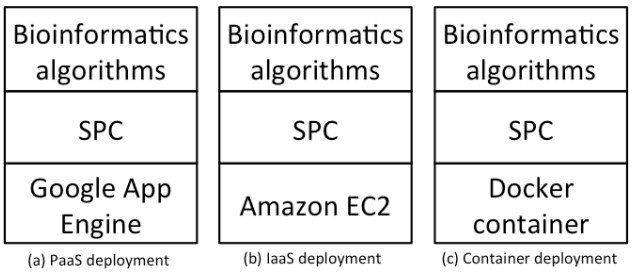

Figure 1. SPC has various options for deployment

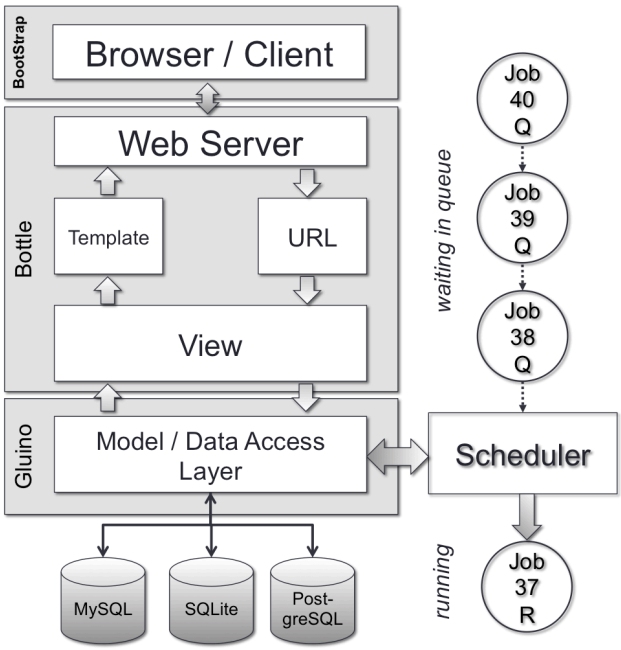

Figure 2. SPC MVT architecture [8]

run on various cloud platforms. For example, running on Google App Engine (GAE) requires using their proprietary Google Datastore database system. The DAL will automatically translate to the language that Google uses for their Datastore, without having to code modifications.

The primary benefit of using an MVT approach is that we can easily generate and store the run-time parameters required for the simulation, and also easily recall them later when needed. Whenever a user submits a new case, a new directory and a universally unique identifier (UUID) is assigned to that case. An input file is written to that new directory. A template engine is used to mix the stored inputs with the HTML to generate the input file view, which is described in more detail in [8]. This architecture allows the user to easily to load either default run parameters, or parameters from any simulations already run.

In this section, we will go through a typical usage scenario.

A. *Starting the run*

The first step in using any software on SPC is entering the run parameters. Once an app has been ported to run on SPC, one can enter the run parameters.

B. *Monitoring the output*

While the simulation is running, the user may monitor the output, or view plots that are updated.

C. *Plotting the results*

SPC can interact with a number of plotting packages, that are either JavaScript-based such as Flot, Plotly, d3.js, and Python-based plotting packages such as Matplotlib. Thus with the interface, one can setup some quite sophisticated plots, without having to learn how to program in JavaScript or Python.

D. *File and Case management*

Users can easily browse through files, zip files, view files, delete files, "diff" (compare) multiple cases for how they differ, etc.

E. *Sharing and Collaboration*

While each user maintains their own cases, they can be easily shared with others on the same cloud server.

F. *Scaling up*

While using a framework such as the Message Passing Interface (MPI) may be useful for scaling up on a cluster, it is difficult to achieve good scaling performance across virtual machines, unless one can provision an entire cluster in the cloud (Guillaume Pierre, personal communications, July 2011). Therefore, we explore other more modern ways to scale our platform, using cloud bursting and Docker containers on Kubernetes.

1) **Cloud bursting**. SPC uses the `boto` Python library to integrate with Amazon Web Services (AWS). Users may register their AWS instances with SPC. This allows one to do cloud bursting, that is, run on a dedicated small machine and when needed burst to a larger machine in the cloud.

For our experiments, we bought a reserved instance of a t2.medium machine (a laptop equivalent machine – 2 vCPUs 4GB RAM) for about $300/year that runs all the time, and when we need a larger run, we startup a m4.4xlarge virtual machine (16 vCPUS 64 GB RAM) which currently costs $0.82/hour. We use Amazon CloudWatch to monitor the m4.4xlarge machine and automatically shut it down when the CPU percent usage drops below 10% over the period of an hour. Using this strategy we are able to have access to a large machine on demand and yet do it at a very reasonable cost. Figure 3 shows the screen where one can start and shutdown m4.4xlarge machine.

Figure 3. AWS Startup/Shutdown view

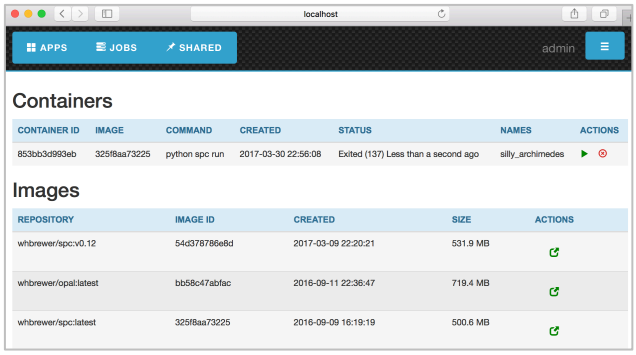

Figure 4. Docker container view

2) **Docker on Kubernetes**. Docker containers have quickly become a popular way of distributing applications. SPC integrates with Docker for (a) distributing a working system, (b) remote workers, and also has an integrated container management system as shown in Figure 4.

In the case of distributing a fully working SPC system with running apps, after installing Docker software, one need only run the following commands to get SPC working:

- docker pull whbrewer/spc
- docker run –d –p 8580:8580 whbrewer/spc:latest
- Open browser to localhost:8580

In the second case of using Docker containers, containers can be used just for running the jobs on the server backend. To implement this approach, SPC requires setting of a `remote_worker_url` parameter in the config.py configuration file. By utilizing this approach, it becomes possible to build a scalable system using a container orchestration system such as Google's Kubernetes (kubernetes.io). In this case, one would push the container to Google's container repository gcr.io. Then, one could start the container using the following command:

```
kubectl run spc-cluster \
--image=gcr.io/myproj/spc:latest  \
--port=8581
```

Once a container is running on Google's Cloud Platform, it can be very easily scaled up or down by first creating a load balancer:

```
kubectl expose deployment spc-cluster \
        --type ="LoadBalancer"
```

Then, at anytime the application can be scaled up or down, so for example, the following command will create four running containers using the same Docker image:

```
kubectl scale deployment \
 spc-cluster --replicas=4
```

Figure 5 shows this kind scenario. SPC runs an interface on Google App Engine (GAE), which itself auto-scales, then schedules jobs to containers. Kubernetes runs the containers on pods, which themselves are easily scaled using the aforementioned command. Each pod can have its own load balancer, which scales incoming jobs on the containers using a round robin policy. Depending on how one builds their system, it is possible to make different pods for different kinds of work. So, for example, we may have a pod with multiple containers running BLAST, and another pod with multiple containers running Phylogenetics software.

Finally, by using the Python docker-py module, SPC can communicate directly with the Docker engine to manage Docker images and start, stop, and delete running containers as shown in the following Figure 4.

III. Results

In this section, we will introduce a number of bioinformatics software packages, which are being used for either teaching or research. Generally, the software used to teach concepts are much simpler than using the production-level software, although some software packages are used in the classroom and for research:

- DNA analysis (*teach*)
- Parsimony method (*teach*)
- Sequence alignment algorithms
  - Smith-Waterman (*teach*)
  - Needleman-Wunsch (*teach*)
  - BLAST (*research/teach*)
- Population genetics:
  - FPG (*teach*)
  - Nemo (*research*)
  - Mendel's Accountant (*research/teach*)
  - Forsim (*research*)

A. DNA analysis

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

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

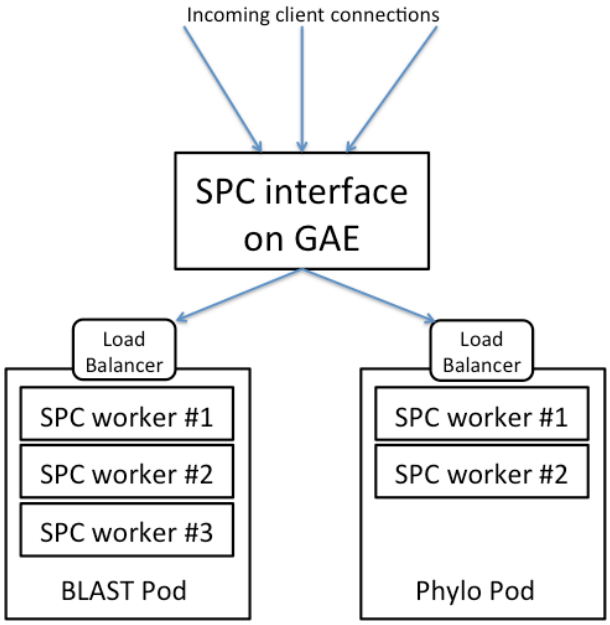

Figure 5. Scaling scenario using Google App Engine, Kubernetes, and Docker containers

- Codon adaption index (CAI)

Usage of this app if further described in [8].

B. Sequence Alignment

In our Bioinformatics course, we teach the Needleman-Wunsch global alignment [9], Smith-Waterman local alignment [10], and rapid alignment methods such as BLAST [11]. We have implemented our own versions of these by forking an open source project https://github.com/alevchuk/pairwise-alignment-in-python and modifying it to run on SPC. Figure 6 shows the global and local alignments of two input sequences. We are in the process of porting BLAST to work with SPC.

C. Phylogenetics software

Phylogenetics software may be used to predict probable ancestor-descendant relationships among a given set of species. In our Bioinformatics course, we teach different methods of constructing phylogenetic trees using parsimony, distance, and likelihood-based methods. We built our own Python-based phylogenetics software that works for a limited number of taxa, and computes the most parsimonious tree. The students learn the algorithm in the class, and then use our software to reinforce what they have learned in class, and try out different scenarios. To plot the most parsimonious tree, we implemented d3.js JavaScript-based plotting software as is shown in Figure 7.

D. Population Genetics

Over the past fifteen years, many forward-time population genetics have been developed. Carvajal-Rodríguez provides a thorough survey of 15 such simulators in [12]. From this list, we chose four population simulators to integrate with SPC: FPG [13], Nemo [14], Mendel's Accountant [15], and Forsim [16]. Nowadays, there are

over 100 listed population genetics simulators given at the master list here:

https://popmodels.cancercontrol.cancer.gov/gsr/packages

Although there have been several review papers to give an overview of these recent simulators, there has not been a convenient way for scientists to easily try out different simulators without downloading and installing each one on their own computer. Depending on the software, installation may range from fairly trivial to very complex. Each of the codes has it's own learning curve. By using a standardized web interface, the user experience of running each code becomes standardized; so that once you know how to use one code there is a much smaller learning curve to run another code. Therefore, we have designed a web-based framework for easily uploading population genetics codes to run and do analysis in the cloud.

By doing this kind of research, we are able to review the codes with more depth than would be possible in a survey paper. For example, we can compare the codes using similar input parameters and test how well they compare against one another. We can determine which codes are well suited for modeling certain biological functions, and which codes do a better job of modeling other aspects. For example, we can run the various codes for similar input conditions and compare the results.

We have done these types of comparisons using the following population genetics simulators. We could only evaluate software that is open source. Figure 8 shows the four simulators installed and running on SPC. The user need only select which ones they want to use to do their simulations.

1) **FPG** stands for Forward Population Genetic simulation. It assumes a constant population size and can model natural selection, recombination, and migration. The model uses an infinite sites model and uses an equal effect for each mutation.

2) **Nemo2** is a flexible framework for simulating various types of scenarios undergoing evolution. It can simulate genes undergoing a variety of different population models [14]. Nemo provides a series of life-cycle events (LCE) such as: breeding, aging, extinction that can be attached to individual traits.

3) **Mendel's Accountant** is a forward-time population genetics simulation program, which models genetic change over time. Mendel's Accountant primarily focuses on biological realism in studying natural selection of human populations, but has also been extended to work for viruses (e.g. [17]), as well as bacterial genomes. Mendel's Accountant has also been used to study a number of problems such as genetic load [18], nearly neutral mutations accumulation, and waiting times for fixation of multiple co-dependent mutations [19].

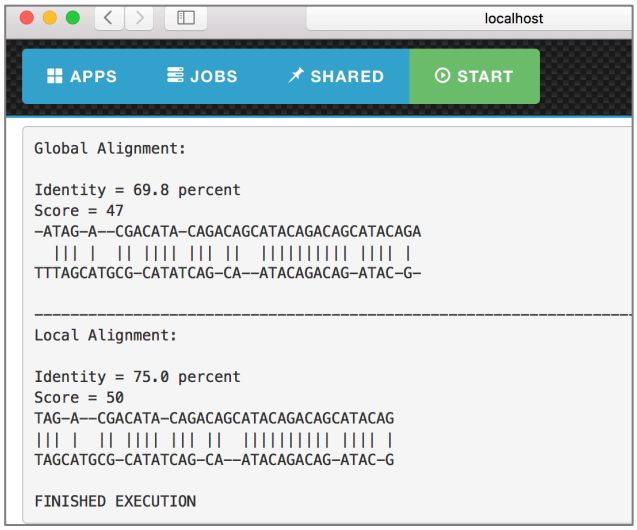

Figure 6. Seqal does both global and local alignment of two sequences

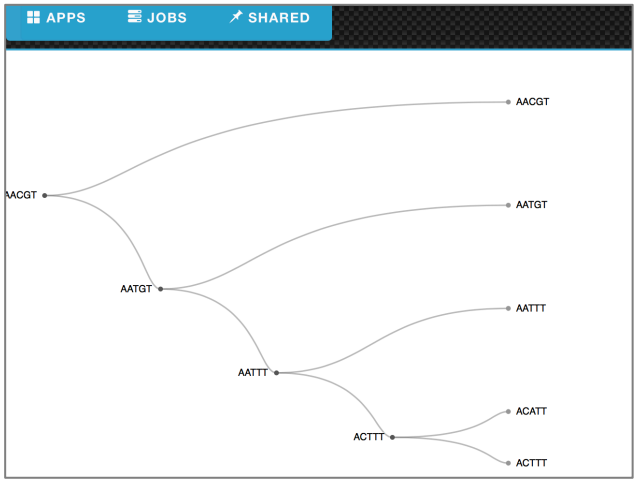

Figure 7. Phylogenetic analysis of five species plotted using d3.js

4) **Forsim** is a very flexible and general framework for studying complex traits in populations [16]. It was written in C++ with Ruby-based wrapper scripts so that it can be easily extended to handle a variety of different scenarios. It outputs many different types of data files that can be further analyzed using R scripts for computing different types of statistics, e.g. linkage disequilibrium.

E. Extensions

In addition to hosting bioinformatics, we have been developing a number of extensions to SPC to make it more useful, both for research and for teaching. So, we have built specialized apps that act as extensions, such as:

- *Assessments* – ability to give quizzes. A *grader* app is used to take the user's inputs and hashed (encrypted) answers and returns a grade. Figure 9 shows an example quiz on using the Mendel's Accountant app.
- *App benchmarking* – there is a special app that is used for comparing and plotting data from two different apps, called simply *plot_two_apps*. Figure 10 shows comparison of fitness distribution between Mendel's Accountant and

Nemo for population size of 1000 and just 100 generations.

- *Workflows* – now it is possible to create a workflow, whereby one may run one app to generate data, and a second app to analyze the data. For example, we are outputting a Variant Call Format (VCF) file from Mendel's Accountant containing various alleles, and then can use VCFtools [20] to do further computation of allele statistics, so that we can compare results with the 1000 Genomes Project [21].

In addition to these extensions, we may also easily embed videos into various apps, thereby effectively being able to build a full-fledged MOOC system that allows one to watch an instructional video, run simulations, and take a quiz to assess the performance, all from within a single framework.

IV. Conclusions

In this paper, we have discussed our work to develop a cloud platform for running bioinformatics algorithms in the cloud. The platform is shown to be very versatile to handle various types of cloud deployments, and can easily work with different types of bioinformatics algorithms that generate various types of data. SPC is open source and the various bioinformatics packages can be freely downloaded at https://github.com/whbrewer.

Acknowledgment

The authors would like to thank FMS Foundation for partial financial support for this study.

References

[1] Kasson, Peter M. "Computational biology in the cloud: methods and new insights from computing at scale." *Pacific Symposium on Biocomputing*. 2013.

[2] Eisenstein, Michael. "Big data: the power of petabytes." *Nature* 527.7576 (2015): S2-S4.

[3] Dudley, Joel T., and Atul J. Butte. "Reproducible in silico research in the era of cloud computing." *Nature biotechnology* 28.11 (2010): 1181.

[4] Schatz, Michael C., Ben Langmead, and Steven L. Salzberg. "Cloud computing and the DNA data race." *Nature biotechnology* 28.7 (2010): 691.

[5] Priyam, Anurag, et al. "Sequenceserver: a modern graphical user interface for custom BLAST databases." *Biorxiv* (2015): 033142.

[6] McElvery, Raleigh. "Letting the knowledge flow: Firehose, FireBrowse & FireCloud" *Broadminded Blog*, 13 July 2015, https://www.broadinstitute.org/blog/letting-knowledge-flow-firehose-firebrowse-firecloud. Accessed 8 March 2017.

[7] Deng, Mario, et al. "Web-TCGA: an online platform for integrated analysis of molecular cancer data sets." *BMC bioinformatics* 17.1 (2016): 72.

[8] Brewer, Wesley, William Scott, and John Sanford. "An Integrated Cloud Platform for Rapid Interface Generation, Job Scheduling, Monitoring, Plotting, and Case Management of Scientific Applications." *2015 International Conference on Cloud Computing Research and Innovation (ICCCRI)*. IEEE Press, 2015.

[9] Needleman, Saul B., and Christian D. Wunsch. "A general method applicable to the search for similarities in the amino acid sequence of two proteins." *Journal of molecular biology 48.3 (1970): 443-453.*

[10] Smith, Temple F., and Michael S. Waterman. "Identification of common molecular subsequences." *Journal of molecular biology* 147.1 (1981): 195-197.

[11] Altschul, S.F., et al. "Basic local alignment search tool." *Journal of molecular biology* 215.3 (1990): 403-410.

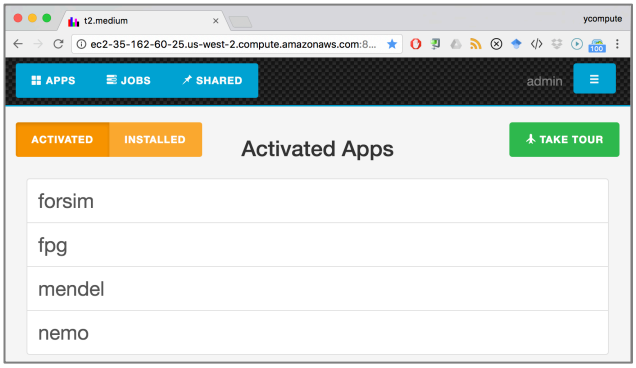

Figure 8. Intro screen to SPC showing the four installed population genetics simulators.

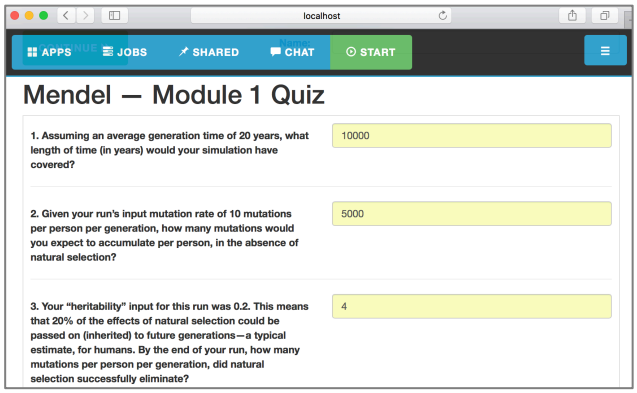

Figure 9. Showing sample quiz

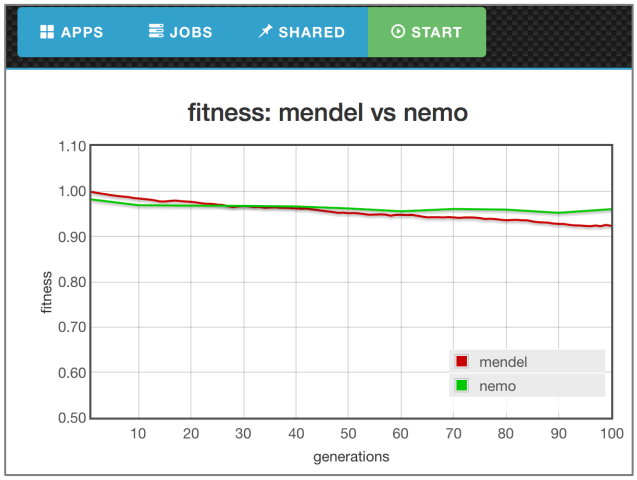

Figure 10. SPC extension is used to compare results from multiple apps

[12] Carvajal-Rodríguez, Antonio. "Simulation of genes and genomes forward in time." *Current genomics* 11.1 (2010): 58-61.

[13] Hey, Jody. "FPG–A computer program for forward population genetic simulation." *Web document: available at http://lifesci. rutgers. edu/heylab/HeylabSoftware. htm# FPG (last accessed date July 14, 2008)* (2004).

[14] Guillaume, Frédéric, and Jacques Rougemont. "Nemo: an evolutionary and population genetics programming framework." *Bioinformatics* 22.20 (2006): 2556-2557.

[15] Sanford, J., Baumgardner, J., Brewer, W., Gibson, P., & ReMine, W. "Mendel's Accountant: A biologically realistic forward-time population genetics program." *Scalable Computing: Practice and Experience* 8.2 (2001).

[16] Lambert, Brian W., Joseph D. Terwilliger, and Kenneth M. Weiss. "ForSim: a tool for exploring the genetic architecture of complex

traits with controlled truth." *Bioinformatics* 24.16 (2008): 1821-1822.

[17] Brewer, Wesley H., Franzine D. Smith, and J. Sanford. "Information loss: potential for accelerating natural genetic attenuation of RNA viruses." *Biological Information–New Perspectives. London: World Scientific* (2013): 369-84.

[18] Sanford, John, et al. "Using computer simulation to understand mutation accumulation dynamics and genetic load." *International Conference on Computational Science*. Springer Berlin Heidelberg, 2007.

[19] Sanford, John, Wesley Brewer, Franzine Smith, and John Baumgardner. "The waiting time problem in a model hominin population." *Theoretical Biology and Medical Modelling* 12, no. 1 (2015): 18.

[20] Danecek, Petr, et al. "The variant call format and VCFtools." *Bioinformatics* 27.15 (2011): 2156-2158.

[21] 1000 Genomes Project Consortium. "An integrated map of genetic variation from 1,092 human genomes." *Nature* 491.7422 (2012): 56-65.