# Sage: Creating a Viable Free Open Source Alternative to Magma, Maple, Mathematica and Matlab

William Stein

## 1 Principal Investigators (PI) full name and contact information

William Stein, `wstein@gmail.com`, `http://wstein.org`, 206-419-0925
Department of Mathematics, University of Washington
Box 354350, Seattle, WA 98195-4350

## 2 Research abstract and goals

The mission of the Sage mathematics software project (`http://www.sagemath.org`), which the PI started in 2005, is to create a viable free open source alternative to Magma, Maple, Mathematica, and Matlab, for research mathematics, industrial applications and education. At its core, Sage is a very large Python library that leverages many of the best established open source mathematics software packages. After 7 years of hard work by several hundred people, Sage is now extremely powerful and full featured, but is difficult or impossible to install on many platforms (phones, Microsoft Windows, etc.). The primary goal of this proposal is to make Sage efficiently available over the web to potentially one hundred thousand simultaneous users by implementing all user and process management on GAE (actual Sage processes will run on some other cloud computing infrastructure in virtual machines, since Sage itself cannot run directly on GAE, as it uses a lot of native compiled code). The PI and his collaborators have already written web applications such as `http://www.sagenb.org` and `http://aleph.sagemath.org/`, but these run on relatively limited hardware resources purchased using National Science Foundation (NSF) grants in 2009, and are not implemented in a sufficiently scalable way. The secondary goal of this proposal is to generate significant revenue (from advertising and "pro accounts") that can be used to support development of Sage at a level that is an order of magnitude larger than what has so far been possible with grants, contracts, donations and prizes.

## 3 Project description with project plan and timeline

The PI directs the Sage project and is the main author of the current Sage notebook, has taught six classes (see `http://wstein.org/courses/`) at University of Washington (UW) about Sage, and has run dozens of workshops on Sage in which he has observed people use the Sage notebook on a wide range of devices. He has amply witnessed how frustrating the current non-scalable Sage web application can be.

Since 2007, the PI has hosted the frontend `http://sagenb.org` on his server at UW, which is *way* over capacity leading to a poor user experience (the demand is there with nearly 100,000 accounts) and it needs to be re-written atop a scalable platform to truly serve the needs of the community.

The PI has extensively prepared for this project by working fulltime this quarter supported by internal funding at UW. He has co-authored some test applications using app engine, including `https://github.com/williamstein/simplesage_gae`, and read cover to cover the book *Programming Google App Engine* (2nd Edition) by Dan Sanderson, and also recently read or re-read numerous other books and source code of the new IPython web-based notebook.

## 3.1 Architecture

Based on extensive testing, the PI is using the following architecture for the new Sage Workspace Server.

1. **Compute Machines:** These are virtual machines running Sage, probably on Amazon EC2. These machines would be reset every few hours to a default snapshot state. They run:

   - **Backend processes:** Python processes that run the TornadIO2 Socket.IO server (which requires the tornado and tornadio2 Python libraries). These will communicate *directly* with end user browsers using WebSocket when available, and falling back to flash or long polling otherwise. The roundtrip time on localhost for submitting a simple computation via Javascript (using Google Chome), getting the result back, then displaying it, are well below 1 millisecond, which feels very snappy indeed.
   - **Process manager:** Each compute machine will run one process manager, which is a multithreaded flask webserver that only listens to the GAE application defined below. When started it registers itself with GAE. When queried (triggered by the GAE taskqueue), the process manager reports its load status. When the GAE application requests a new session for a given user, it starts a new process, and reports back the port of the newly started process. The GAE application then relays the compute machine's address to the web browser client, which makes a Socket.IO connection to that backend process. All communication between the backend process and the web browser client happens directly via Socket.IO, except sending SIGINT and SIGKILL signals, which happens via the process manager. Also, files (which are part of a session) are transferred between the GAE application via the process manager.

2. **Frontend – the GAE application:** This manages storage of user worksheets, command line sessions, data files, etc. (supporting Google Drive), and serves all static and templated Javascript and HTML content. The datastore will track which worksheets (etc.) are published, shared, etc. In addition, when new compute machines are turned on, they register themselves with the frontend. The frontend periodically (using taskqueues as needed), determines the status of compute machines, in order to determine how to allocate them to users.

3. **Clients:** These are Javascript programs that run in a web browser. They are implemented using Socket.IO, jQuery, and jQuery-mobile for mobile devices, HTML5/CSS3, and WebGL for 3d graphics. The core of the client is a Javascript library that interacts directly with backend processes using Socket.IO; this library must be as robust, fast and well tested as possible, since it will be critical to building the clients. A worksheet (or command line session, etc.) will periodically get saved to Google Drive (or other clients such as DropBox). Based on past experience with the Sage notebook, the PI expects to spend a large amount of time implementing and polishing the client, and will also implement a basic mobile version using jQuery-mobile and PhoneGap.

With the above architecture, it is not too difficult to scale the backend compute machines—simply turn on more of them. However, it is very difficult for *me* to scale the frontend in this design, which is why Google App Engine is the optimal deployment platform for the frontend. The frontend will get an enormous amount of traffic, and it is critical that it always be extremely responsive, since it serves the client Javascript program, and all static user content.

In the longrun, the PI intends to serve advertisements on the frontend. For example, the published worksheets would include relevant ads, and be indexed by Google. Based on current usage of `http://sagenb.org`, I've estimated that this should make a reasonable profit. However, if the PI receives this grant, he will be able to serve everything for a year with limited (or no) advertising, which should help initially in building a large user base. After the grant expires, the PI can crank up advertising and start generating revenue (to support the Sage project). The PI also plans to sell more powerful "pro accounts", which will have less restrictions, e.g., computations run on compute machines that have more cores and memory. In addition, the PI plans to sell site-licensed, branded frontends for universities; this could be on the order of \$1/student and still look like pennies compared to a site-wide Mathematica license; moreover, setting up a university notebook (or going with the public one, or getting students to install Sage themselves) seems to be a major pain point for people wanting to use Sage in their classes.

### 3.2 Timeline

Due to support by UW and the NSF, the PI is able to devote fulltime effort to this project until September 23, 2012, except for 3 weeks of workshops he is organizing. One week is an advanced research workshop funded by NSF mainly for young postdocs, which will use a first version of the software. The other two weeks the PI will run a workshop (SIMUW) on the Riemann Hypothesis for high school students, which is held for 3 hours a day in a computer lab, and will provide a realtime test environment for the software.

1. (deadline: June 12, 2012) Implement usable, tested and robust versions of the core components of the system, with *bare bones* functionality, at least running locally on a single computer. Sage Days 41 on the Sage Notebook starts June 13 (see `http://wiki.sagemath.org/days41`) and will provide an opportunity for feedback and collaboration.
2. (deadline: June 23, 2012) Deploy the new bare bones notebook in time for NSF workshop.
3. (deadline: July 22, 2012) Implement all of the important *features* on top of the core functionality mentioned above. Also, get access to more EC2 (or other) resources, using an academic grant program that Amazon has (also Jeff Barr – "Lead Web Services Evangelist at Amazon.com"– audited some my Sage course at UW, when his son took it). Use in SIMUW workshop.
4. (deadline: August 27, 2012) Based on the experience above, implement additional core functionality, write better performance and correctness testing code, optimize use of App Engine and virtual machine resources, fine tune load balancing, create native mobile clients using PhoneGap and a jQuery-mobile client, and incorporating `three.js` 3d graphics into the notebook. On August 27, announce the application as widely as possible. Implement a strategy to encourage people to switch from `http://sagenb.org` to the new GAE server.
5. (deadline: September 23, 2012) Fine tune everything mentioned above based on observed usage. Ensure new server has as many of the important features of `http://sagenb.org` as possible (but note that sagenb has accumulated a huge number of features over the years).

The above time table is based on my past experience implementing much of the current Sage notebook, and the fact that the PI has already done most of the necessary background reading and experimentation to get up to speed with current open source web technologies.

## 4 Expected outcomes and results

Making Sage widely available has the potential to significantly impact how mathematical research and teaching is done. There are now nearly two hundred scientific publications (see `http://sagemath.org/library-publications.html`) that cite Sage; by making Sage more accessible via the GAE infrastructure, the usage of Sage may increase greatly. Moreover, this project is likely to benefit those high school and college students who take some form of mathematics classes, because Sage provides interactive features that aid in teaching mathematics, and many high schools and colleges cannot afford Mathematica or other commercial licenses, and find installing Sage locally to be too cumbersome.

If you fund this proposal, there is no question the PI will be deploying the frontend part of the application on GAE. Otherwise, due to concerns about the GAE cost structure, and the fact that Amazon does grant compute resources to academics, the PI may have to deploy the entire system on EC2.

## 5 People at Google who are familiar with your professional work

- Craig Citro (Seattle) – he initially suggested the PI use GAE (was also my postdoc)
- Robert Bradshaw (Seattle) – my Ph.D. student
- Rado Kirov (Mountain View) – one of the main developers of the Sage notebook
- Christopher Swenson (Washington, DC) – Sage contributor