1. Multi-resource Perfume resource

Perfume uses resources recorded in the log to improve model quality. Perfume focuses on a single resource at a time. However, sometimes, it makes sense to consider a combination of resources. For example, if the log contains packet size and time information, one might want to build a model using speed (packet size / time) data. Perfume cannot do this right now.

The project's goal is to allow the user to (1) specify how to parse multiple resources from the log (by extending the existing regular expression slightly), and (2) to specify a formula for using the parsed resources to create a new resource (e.g., speed = packet size / time). Then, the tool will take a log, parse it, compute the new resource, and rewrite the log with the new resource inserted.

You can see the Perfume deployment here:
http://perfume.cs.umass.edu/

2. Create a robust and flexible Java-based vector clock time-stamping library.
We read about ShiVector, a java library that automatically inserts vector clock timestamps into logs. ShiVector uses AspectJ, which is great when you can use Aspects and the library's default instrumentation is sufficient (e.g., you are using
plan Java sockets). But it doesn't work well if you can't use Aspects (e.g., can't compile your code using the AspectJ compiler), or if you are using a networking library, or if you need to access the clocks yourself (i.e., you don't want transparent clocks). The idea would be that, unlike ShiVector which doesn't force any modifications to the client code, this library is something the client uses explicitly (i.e., makes calls into your library).

Some starting design constraints for this library:
- Usable with any existing Java library
- Can integrate well into existing logging libraries like log4j, and java.util.logging
- Provides a facade for logging libraries, like Apache Commons Logging or SLF4J (supporting both would be best), see this question:
  http://stackoverflow.com/questions/9740569/logger-for-java-library
- Including support for protocol buffers: https://code.google.com/p/protobuf/

The library needs good documentation, a good test suite, and examples of integration with other large projects (to demonstrate that the library works). Of big help is that the vector clock code itself already exists (in many versions). These can serve as great starting points:
ShiVector:
https://bitbucket.org/bestchai/shivector/src/941619a920f3c12de3d15aa0a945391c52765fd9/java/shivectorasp/src/shivector/aspects/VectorClock.java?at=default
Voldemort's vector clock library:
https://code.google.com/p/project-voldemort/source/browse/trunk/src/java/voldemort/versioning/VectorClock.java?r=170
Another project's vector clock library:
http://tud-in4150-fp-ass2b.googlecode.com/svn/trunk/src/in4150/mutex/VectorClock.java

3. Real timestamps for ShiViz DAG view

See these three related issues. The combination of these three make a good project:
https://bitbucket.org/bestchai/shiviz/issue/7/reveal-real-timestamps-on-the-timeline
https://bitbucket.org/bestchai/shiviz/issue/13/showing-a-partial-order-dag-adjusted-for
https://bitbucket.org/bestchai/shiviz/issue/22/navigating-a-long-log-with-scrubbing

The basic idea is that the log may include real timestamps (as opposed to vector-clock timestamps). We can parse these and it would be great to integrate them into the DAG itself. The last issue "navigating-a-long-log-with-scrubbing" is a way of utilizing the integrated timestamps, in other words, it's a feature that uses the timestamps.

The key challenges are:
- Understanding the existing codebase. The code is of very high quality, but it's large and it's all in JavaScript. The student would need JS experience.
- Figuring out how to reconcile conflicting timestamps from different nodes. We can experiment with some simple solutions (e.g., just use the timestamps from one node and ignore the others), but we'd have to understand the best thing to do here.

You can see the ShiViz deployment here:
http://bestchai.bitbucket.org/shiviz/

4. Improve the Synoptic GWT project/deployment.

The Synoptic web deployment, http://synoptic.cs.washington.edu/, can be improved. It uses GWT (Google Web Toolkit), but not the most recent version of it. GWT 2.7 was recently released:
http://googlewebtoolkit.blogspot.ca/2014/11/gwt-27-finalized.html

Key tasks:
- Upgrade GWT to 2.7
- Fix a couple of outstanding issues:
-- Bug fix:
https://code.google.com/p/synoptic/issues/detail?id=394
-- Refactor the client/server interface:
https://code.google.com/p/synoptic/issues/detail?id=70
-- Simple layout improvement:
https://code.google.com/p/synoptic/issues/detail?id=65
-- Simplify display:
https://code.google.com/p/synoptic/issues/detail?id=395

There are also several new features that would be great to add:
- https://code.google.com/p/synoptic/issues/detail?id=98
- https://code.google.com/p/synoptic/issues/detail?id=84
- https://code.google.com/p/synoptic/issues/detail?id=71

5. Stateful Synoptic

Synoptic ignores the variable values in its executions. It would be great to add to the synoptic models the values that variables took on at each state. This write up by a former student, Roykrong, describes the ideas of this project:
http://people.cs.umass.edu/~brun/class/2015Spring/CS320/StatefulSynoptic.pdf
Your job would be to build an implementation of this tool.

6. ShiViz improvements
The ShiViz tool has many features that can be added. Take a look at the list of open features here:
https://bitbucket.org/bestchai/shiviz/issues?status=new&status=open
Accomplishing several of these this semester would be a good project.
You can see the ShiViz deployment here:
http://bestchai.bitbucket.org/shiviz/