Problem Statement: Creating Provenance from System Call Traces

March 4, 2013

Abstract

This document gives a high level overview of the research project and its goals, as well as details on the inputs and expected end result.

1 The research project

Science is the art of coming up with questions and their answers, so it's always good to start with the questions. Obviously, there are lots of things we can do with the data, but here are a few things I'm trying to figure out.

1. Given a provenance graph, can we accurately predict what file people will access next?
2. In particular, can we figure out what ONE person will access next? (Personalized prediction)
3. Can we identify some characteristics of file access that are unique to bioinformatics?

What we have right now is a bunch of long term system call traces from systems that have been used for computer science research, and a bioinformatics cluster that hasn't been installed yet. So we're focusing on turning the traces we have into a graph, and then once the cluster comes up, we can worry about getting traces from it. (Also, if we run into snags with the traces we have, we can avoid wasting time later)

2 The data

The data we're working with is from the LASR system. It will look something like the following:

```
0 RESTART (scratch) at 962148771 Tue Jun 27 16:32:51 2000
28 RESTART (scratch) at 962583623 Sun Jul  2 17:20:23 2000
```

```
56 UID 0 PID 1 ?? A 962559820.187587 unlink("/etc/initrunlvl") = -1 (2)
96 UID 0 PID 1 ?? A 962559820.201366 unlink("/var/log/initrunlvl") = -1 (2)
140 UID 0 PID 10 ?? A 962559821.223185 unlink("/dev/log") = -1 (2)
176 UID 0 PID 10 ?? S 962559821.223242 exit(1280) = 0
```

It's kind of obtuse, but you can see that there's a uid and a pid (user and process). There's a UNIX time stamp (like 962559821.223242), a system call name (like unlink), and then some arguments, and a return value.

3 Provenance graphs

A provenance graph has nodes, edges, and "annotations" (basically extra info). There are two kinds of nodes, processes and files. Edges connect files to processes, or vice versa, and are directed in order of dependency. So, for instance, if a process opens a file and reads it, then the edge goes from the process to the file, because the process "depends" on the file (See Figure 1). For a write, the edge goes the other way, from the file to the process (Figure 2). If a process starts another process, then the child depends on the parent, so the edge goes from the child process to the parent process (Figure 3).

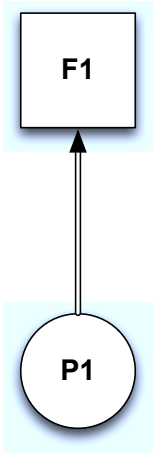

Figure 1: Process reads file.

However, that's not quite as much information as we need for full causality. So we add annotations, in the form of users, time stamps, and versions. When a file is changed, we declare that to be a new version of that file, so we might have results.txt(v1), results.txt(v2), etc. That way we can say which version of a file another file got data from, and we know which processes contributed up to that point. (And it breaks cycles which would otherwise be very confusing!) I have an example in Figure 4.

Example graph:

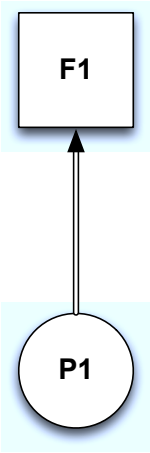

Figure 2: Process writes file.

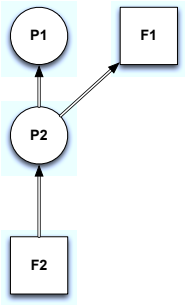

Figure 3: Process starts another process which reads one file, and then writes another.

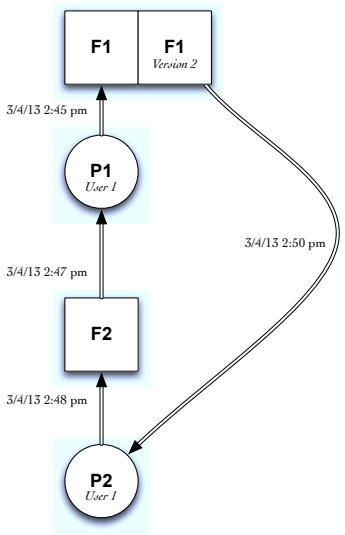

Figure 4: Process reads file.