The Plaz Provenance File System (PPFS)

**Proposal**

Michael Plasmeier Rudolph 10AM

March 1, 2012

# Introduction

A provenance file system is a file system which stores the history and source of files edited on a local computer system. This paper builds upon the basic file system in the early versions of Unix and introduces a provenance file system known as the *Plaz Provenance File System (PPFS)*. In particular, the PPFS introduces another layer, called the log layer, which maintains pointers to past versions and ancestors of the file.

# The Log Layer

The PPFS introduces a new layer called the log layer. This layer is inserted between the file name layer and the inode layer, as shown in Figure 1. The file name layer is modified by redefining the inode number in the directory table to the log entry number.

File Name Layer

Log Layer

Inode

Inode

Inode

Log entry #

Inode #

Figure 1 The Log Layer is inserted between the file name later and the inode layer

The log layer contains a table of information about the history of the file. The table is stored on disk in the same way as the inode layer. The log layer table is stored at the beginning of the disk, in a fixed position on the disc, after the inode table. The table consists of a list of log entries, each pointing to the inode number of a *version* of the file, as shown in Figure 2. A version is created automatically each time the file is saved. The last entry in the inode block contains a reference to the log entry that the file was created from (the *ancestor)*. A bit in each entry designates a row as a version/inode pointer or an ancestor /log layer pointer. If a file was created from scratch (ie using touch) then the last entry will be 0. Since the inode stores the time the file was modified, that information is not stored in the log table. The number of incoming links that was stored in the inode is redefined to count the number of log layer entries pointing to the inode. The log layer table includes a count of the number of incoming links that was traditionally found in the inode, along with a count of children who have that file as its ancestor.

**Log entry 987654  
98765**

**Ancestor**: 00000000

**Inode**: 123456

**Directory /**

|  |  |
| --- | --- |
| File name | Log Entry # |
| File A | 987654 |
|  |  |
|  |  |
|  |  |
|  |  |

**Inode 123456**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

**Inode**: 123457

**Inode 123457**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

Figure 2 File A is created with content and is then edited.

**Log entry 987654**

**(File A)**

**Ancestor**: 000000

**Log entry 987665**

**(File B)**

**Inode**: 123457

**Inode**: 123456

**Ancestor**: 987654

**Inode**: 123458

**Inode**: 123457

Figure 3 File A, from above, is then copied to be File B. File B is then edited. Notice that File B rev 0 shares an inode with file A rev 1.

When provenance information is queried for File B, the log entry for File B will be retrieved. Provenance information will then be recursively queried (to A in this example) until 0 is reached.

## Parts of a File

Provenance information can be stored about the *parts* of a file (for example, the slides in a slide deck or the source files of a compiled file). This information is stored by having multiple ancestors in the log layer, as shown in Figure 4. In this example, the difference between Slide Deck E version 2 and version 1 came from Slide Deck D. The pointer to Slide Deck D would be set by the write\_prov() call. Information about the source in compiled binary files and file archives can be stored in the same way, with multiple ancestor entries at the bottom of the file, with the final entry being 0.

**Log entry 987636**

**(Slide Deck E)**

**Ancestor**: 987352

**Inode**: 126268

**Inode**: 126267

**Ancestor**: 987640

**Inode:** 126269

Slide Deck C

Version 0

Version 1

Version 2

Slide Deck D

Figure 4 Slide Deck E was copied from Slide Deck C with 2 slides; edited; and then a third slide was copied in from Slide Deck D

# Deletion and Thinning

When unlink(filename) is called, the filename to log entry link is removed, and the traditional link count in the log layer entry is decremented. This count will reach 0 if the file is no longer on desk or pointed to as an ancestor. In that case, the log layer entry, as well as all of the versions, is removed.

In order to save space some intermediate versions can be removed according to a thinning schedule. This schedule is user-settable, but the default values are shown in Table 1. The thinning process is accomplished by a “garbage collection”-style program. When versions are thinned, the actual version is removed from the disc, and all references to that version are removed from the log layer. A revision will be kept if other files refer to it in its provenance information. This can be quickly queried by looking at the revision’s inode link count.

|  |  |
| --- | --- |
| Days after revision created | Target number of revisions kept |
| < 7 days | 1 / minute |
| > 7 days and < 30 days | 1 / hour |
| > 30 days and < 1 year | 1 / day |
| > 1 year | 1 / week |

Table 1 Intermediate revisions can be thinned after a certain amount of time after their creation. These are the default values

# Requirements

It is believed that this proposal meets the requirements as set forward in the assignment.