Playback Testing Using Log files

Author
Vijay Upadya, vijayu@microsoft.com

Abstract

Most testers are familiar with record/playback testing. A variation of this approach is log playback testing. Instead of explicitly recording the user actions using a recorder, this technique simply relies on the information in the log file produced by the system under test to play back the scenario. This is extremely useful for reproducing issues reported by customers which are typically hard to reproduce otherwise due to lack of detailed repro steps. This approach also lends itself well to creating data driven tests and can be used to augment the existing test automation.

Testing and investigating failures in a non-deterministic software system can be very costly and painful. In most cases the only source of information to know what code path got exercised are the log files produced by the application. This paper talks about how the information in log files can be used to playback the sequence of actions to test specific code paths and also reproduce bugs. We will look at exploring an extension of this approach and how log files can be used as a source for data driven testing by simply varying the sequencing of actions that are played back. This paper illustrates the technique using real a world example. Finally the paper discusses the benefits and how it can be applied to different test domains.

Biography

Vijay Upadya is a senior software development engineer in test at Microsoft, currently working at the Microsoft main campus in Redmond, Washington. Over the past decade, he has been involved in software development and testing for products ranging from compilers to libraries like Linq To SQL, Silverlight and RIA Services to Windows Live mesh. Vijay primarily focuses on test strategy, test framework and tools and test frameworks for the team.

Vijay has a Master's degree In Systems Analysis and Computer Science from the Mangalore University, India

1. Introduction

Log files are typically used for diagnosing issues reported by users of the product. In most cases log analysis is done manually where developers read the log and try to understand the sequence of events happening during the failure and where it went wrong. For catastrophic failures like crashes the call stack will mostly suffice in figuring out the issue in the product. But when there are functional failures, having a detailed log will go a long way in helping diagnose the issue quickly.

Now we can take it further and design logging with the intent of generating logs for easy auto analysis and use it to actually playback the scenario as it happened in user's machine. In order for this to be successful, the logging format needs to be designed right from the beginning of the product cycle with playback in mind.

Log files are also a great source of information that shed light on how customers are using the product and information contained in them can be sourced as test cases that test teams can then run regularly during their test passes.

In the next sections we will discuss how log files can be used for playback and more exhaustive testing. We will use an example of file synchronization software[i] for the discussion.

2. Information needed for playback

In general log files for the application contain sequence of important events that happen along with information about data. Let us explore characteristics of a good log that is 'playback ready'.

There are two main pieces of information needed for playback

- User intent (i.e., scenario user intended to accomplish)
- Path taken by product in fulfilling this intent (i.e., product's execution flow)

2.1 User Intent

User intent is about capturing what a user intended to do while using the product. It could be a simple task or a series of tasks as part of a larger scenario.

Examples:

- User performing mathematical operations on a series of numbers in a calculator application
- User copying photos from camera to local folder on his PC and syncing it to other PCs using file synchronization software.

A log file needs to provide enough information to make it possible to interpret the user intent. There are two items of interest for capturing user intent

- Timeline of actions performed by user
- Data passed to each of these actions

For simple applications like a calculator, just capturing user intent will probably suffice as the component behaves in a deterministic manner for a given input. For example, user presses 2 + 4= and calculator produces 6. The calculation logic is synchronous and execution flow deterministic. However for non-deterministic software like file synchronization where there are a number of threads executing in an asynchronous manner, and where the order of operations are not guaranteed, just capturing user intent is not enough. We need an additional piece of data – the path taken by the product or execution flow.

2.2 Product Path

Path taken by the product: When a user performs a specific task, the product itself will perform a series of tasks to accomplish the user intent. These tasks are typically implemented in multiple components and some of these could be executed in parallel.

There are two items of interest for capturing product path

- Timeline of states and their transitions while executing tasks
- Information on any concurrent execution of tasks

3. How to capture user intent

In general, in order to capture user intent the product needs to understand general user actions or entry points to the application. This would be happening in any functional product. For example, when a user presses '+' in a calculator application, the application will 'interpret' it as an addition (user intent). In most cases it's just a matter of logging this information at appropriate locations in the product where it 'understands' the user intent. The timeline is also an important piece of data that enables determining the ordering of operations happening in the product and helps identify any concurrent operations happening. The product needs to log the timestamp information indicating when this action occurred along with any relevant data passed.

4. How to capture the product path

This involves thinking through control flow of the product while realizing the user intent and would involve logging different state changes and their transitions. Again here timestamp information needs to be logged along with data. More on this is discussed in the next section with an example.

5. How to playback

There are two things that need to be played back - one is the user intent and the other is the control flow of the application. For simple applications, just playing back the user intent (using the same data and in the same order) would help in replaying the scenario and reproducing the bugs. However for more complex products that are non-deterministic in nature (meaning, for the same scenario it could take a slightly different code path due to its asynchronous nature) the product path also needs to be 'played back' so that it converges on the path that caused failure in the user scenario.

Let's take an example of file synchronization software. There are three main operations that users of this application typically perform – Creation, Updates and Deletion (CUD) of files and folders. Now these can happen in any order at any time on any number of files on any number of devices that are enabled for sync. The product need to understand each of these main user gestures (CUD) and it's easy for it to log when it happens.

Looking into the control flow of typical file synchronization software while performing these CUD operations reveals that it comprises of the following sub-operations

- *Scanning* for detecting changes,
- *Uploading* (file to cloud storage),
- *Downloading* (file on another machine) and
- *Realizing* (copying) the file downloaded to right folder location so that the file appears on disk on other machine.

Some of these operations can happen in parallel asynchronously (for example, uploading and downloading).

Let's say a user reported a 'failure to sync' issue and shared the log file which looks something like this (only relevant portion shown for illustration purposes) with the log format 'TIME_STAMP' 'TASK' 'Data'

```
06-14-2011 00:08:46.507 SCANNER: thread for 'C:\Users\vijayu\SyncFolder' started
06-14-2011 00:08:46.617 SCANNER: FILE_ACTION_ADDED for 'C:\Users\vijayu\SkyDrive\ myPhoto1.jpg'
06-14-2011 00:08:48.703 SCANNER: thread finished scanning
..
06-14-2011 00:08:49.826 DOWNLOADIN file 'myPhoto2.jpg'
06-14-2011 00:08:49.827 UPLOADING file 'C:\Users\vijayu\SyncFolder\myPhoto1.jpg'
..
06-14-2011 00:08:51.626 DOWNLOADIN file 'myPhoto2.jpg'
06-14-2011 00:08:51.629 UPLOADING file 'C:\Users\vijayu\SyncFolder\myPhoto1.jpg'
..
06-14-2011 00:08:53.322 DOWNLOADIN file 'myPhoto2.jpg'
06-14-2011 00:08:53.327 UPLOADING file 'C:\Users\vijayu\SyncFolder\myPhoto1.jpg'
```

From the above log snippet it's easy to figure out that a new file myPhoto1.JPG was added, picked up by scanner and it's trying to upload it. Also it's trying to download a different file myPhoto2.JPG (on a different thread) at around the same time. Also this looks like the point of failure as log contains repeated section of the above uploading/downloading sequences. This might be an indication of the application getting into some kind of infinite loop and the issue surfaces only when upload and download happens simultaneously. This can happen when user has added files on both machine1 and machine2 and each is making its way onto the other machine. On the machine where failure is happening, the upload and download are occurring simultaneously resulting in some kind of a failure.

So to summarize we have following information:

- User intent: Addition of file on two machines and syncing to all machines
- Product path: Scanning completed, Simultaneous download and upload task started

Now for playing it back, the playback tool needs to detect these two pieces of information. From the line on scanning along with file name it's straightforward to interpret that the user added a new file myPhoto1.jpg and this condition can easily be reproduced by adding a new file to the path specified.

The lines in log on downloading tasks along with file names indicate that there was newly added file on the other machine that it's trying to download to this machine (where failure occurred). So to simulate this condition, the tool needs to add a file to a different machine to the path specified.

Examining the timestamp for upload and download indicates that these two tasks are happening (nearly) simultaneously. Putting it all together, these are the steps playback tool need to perform

- Create file myPhoto1.jpg in machine1
- Create file myPhoto2.jpg in machine2
- Wait for download to start for file myPhoto2.jpg
- As soon as download starts, also start uploading myPhoto1.jpg

User intent actions can easily be reproduced by adding a new file to the path specified on machine1 and machine2. However controlling the flow to ensure download and upload occurs simultaneously is a little tricky.

In order to accomplish this control over flow of execution our team used the detours[ii] library. In a nutshell the detours library enables hijacking function calls in the product and gives test code a chance to control its execution timings. The way this is used in the example case for file synchronization mentioned above is:

- Create a test binary that detours upload and download functions by providing a detour function
- Load this test binary to the product process

- In the upload detour function wait until download is ready to begin
- In the download detour function wait until upload is ready to begin
- When the condition is met, continue the execution

6. How it can be used for test generation

The test team could use this playback technique to pro-actively identify issues before releasing the product. This can be achieved by taking an existing log for a given scenario (say produced by functional tests), interpreting the log to identify different states in the product and simulating all possible control flow conditions.

For example, in the above file sync case, on creating a file to sync folder we noted that product goes through four states. From this we can come up with the following three cases that can occur simultaneously:

- Download during Scanning
- Download during Uploading
- Download during Realizing

The same playback tool can then be used to simulate the above three conditions and test the products behavior.

7. Conclusion

The log playback has the potential to make reproduction of issues less painful and cheaper. It does require initial investment to come up with the infrastructure for playback, but once it's in place, it can rapidly enable test teams to test different conditions/code paths the product could take and also to reproduce issues reported by the users.

8. Reference

[i] http://en.wikipedia.org/wiki/File_synchronization

[ii] Galen Hunt and Doug Brubacher. *Detours: Binary Interception of Win32 Functions*,