Erlang On The Cloud

Dropbox and Concurrency

Kristen Rebello
BSc Computer Science, CSD2600
Middlesex University
Hendon, London, NW4 4BT
KR523@live.mdx.ac.uk

***Abstract*— Two kids have access to a shared toy. If both the kids try and play with it, at the same time, the toy may end up broken. This would be because, the kids are concurrently trying to access the toy, and one is interfering with the other. The solution would be to have the kids play with the toy, one after the other. In this paper, we consider the concurrency issues with cloud services. More specifically, we will look at Dropbox. We will also discuss the concurrent execution and interleaving of processes as the processes interact, and how Erlang can be used to implement this scenario and help find a solution to this problem. We will investigate a potential solution, mutual exclusion (Mutex), and how it works in Erlang.**

Keywords—concurrency, Erlang, Dropbox, interference, Mutex, semaphore, Cloud

I. Introduction And Overview

If you have access to more than one device, like a smartphone and a PC, then you would know the troubles of when you have to share files across them. There is a solution to this - 'Cloud Computing', also known as 'The Cloud'. The user's information and files are stored on servers, and any device that's connected to the internet can access these files. "You can't see the cloud in the sky, but it's definitely got a silver lining [1]."

The cloud is also widely used in the business environment, where employees may share and work on the same files. It lets you upload files onto the server and lets any device you want synchronize with it, to access any of those files.

File synchronization is the process of ensuring that any computer files or resources in two or more areas, are updated based on certain guidelines. Services like Dropbox, Google Drive and OneDrive are examples of file sync services and are becoming a necessity, be it for businesses, schools or personal use. Hundreds of millions of people entrust their data to these services every day [2].

The problem occurs when we must allow more than one person to edit the files at the same time. In this paper, we will discuss this problem - how the concurrent use of a shared resource, could be a source of indeterminate results and could result in interference and issues such as deadlock and starvation of the resource.

Erlang is a concurrent programming language – this means that any parallel processes can be programmed directly in Erlang [3], which would assist in the problem of having two users access a single file, to read and write. And because Erlang is a single assignment language, i.e. variables are immutable, they don't need concurrency protection [4].

In this paper, I will first provide a brief introduction to Erlang and its concepts, terms and the functions involved in multi-processing (Section II). Then, there will be a detailed description of the problem with concurrency and Dropbox, which will include formal descriptions and presentations of the scenario (Section III). A possible approach to the scenario and a solution to the problem, mutex, will be described in detail, also giving formal representations of the machine (Section IV). This will be followed by a short discussion of the advantages and disadvantages of Mutex (Section V). A brief conclusion on mutex will be given (Section VI).

II. Background

In this section, I will briefly describe the various terminologies and functions that are used in the rest of this paper, along with a brief introduction in the Erlang concepts.

A. Concurrency And Interference

Concurrency, in this context, is the ability of having two or more processes running at the same time. This concept may be similar to parallel processing, but with concurrent programming, there is the opportunity of having numerous independent jobs doing different tasks at once, rather than executing an identical job [5].

The whole point of concurrent programming is dealing with the interference between threads or processes. If processes access a shared resource, sometimes the result state or output of the shared resource can become incorrect due to a destructive update, caused by the arbitrary interleaving of actions. This is known as interference [6].

B. *Erlang And Concurrency*

Erlang is a parallel programming language. To control a set of parallel activities, Erlang has primitives for multiprocessing: 'spawn' starts the execution of a parallel process, 'send' sends a message to a process and 'receive' receives a message from a process [3].

spawn/3 starts the execution of a parallel process with the name of the module, the behavior and the arguments of the behavior, and return the process identifier of the process. For example, spawn(dropbox, client, [""]) causes dropbox:client("") to be executed in parallel.

The syntax Pid ! Message is used to send a message. Pid (Process identifier) should evaluate to a process identifier and Message is the message which is to be sent to the process. All arguments are evaluated before sending.

receive has the following syntax:

```
receive
      Msg1 -> ... ;
      Msg2 -> ... ; ...
end.
```

Each of the processes have a mailbox. When a process receives multiple messages, they are stored in the order they are received, and actions are evaluated in the order they are programmed. Once the action of a message is evaluated, the message is removed from the process's mailbox. Any message that is not matched, will remain in the mailbox.

register/2, register(name, Pid) associates the name, of type atom, with the Pid. "name" can then be used instead of the Pid, after this function has been run.

For the Dropbox application, the function server/1 acts as a server on which the file would be stored. That is the file onto which we will write and read from. There are three other functions we will look at:

read/0 takes in no arguments and returns the contents of the file.

write/1 takes in a string and writes it to the file. It returns a tuple with an atom 'write' and the string.

```
4> dropbox:write("er").
{write, "er"}
5> dropbox:write("lang").
{write, "lang"}
```

write/1 overwrites the empty string it starts with every time you use dropbox:write(Str).

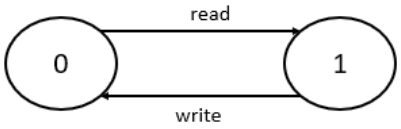
```
6> dropbox:read().
"lang"
```

upload/1 takes in a string and adds it to the file. It reads what the file already has stored in it and assigns it to the variable OldFile. The string which is input is added to OldFile and the new string is assigned to the variable NewFile. It then writes NewFile onto the server.

```
4> dropbox:upload("er").
{write, "er"}
5> dropbox:upload("lang").
{write, "erlang"}
```

This is a basic Dropbox program wherein a single user can read and write from a file.

UPLOAD = (write -> read -> UPLOAD).

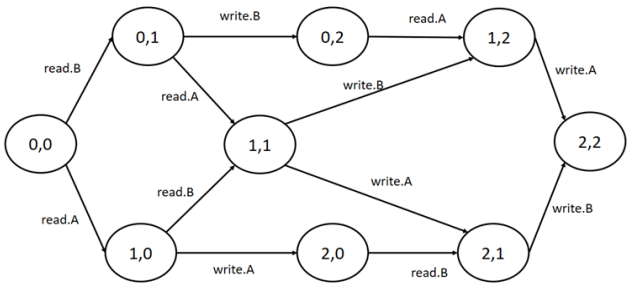


III. Problem Description

The problem arises when there is more than one person accessing the same file (in this case, trying to access the server). If multiple people have access to the same file, it can cause an overlapping and interference, which can cause indeterminacy of the output.

When you spawn two processes at the same time, the two processes start interfering with each other and give a different output from the expected output.

DROPBOX = CLIENT_A || CLIENT_B



Above is an FSM that presents all possible traces, including the successful traces. The incorrect outputs we get are because the first process interferes with the second, and the trace of the machine could be as follows:

read.A->read.B->write.A->write.B

read.B->read.A->write.A->write.B

Client A reads an empty file. Client B then reads the file (It is still empty). Client A then writes "er" to the empty file it has read. Client B writes "lang" to the empty file it had read before. These actions interfere and so Client A has been overwritten. Other possible traces, where Client A overwrites Client B, include:

read.A->read.B->write.B->write.A

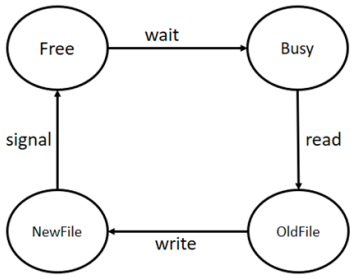
read.B->read.A->write.B->write.A

One process starts a little after the other, so the screen will display the output of the last process to finish. Actions cannot really happen in parallel. They are processed in sequential parallelism.

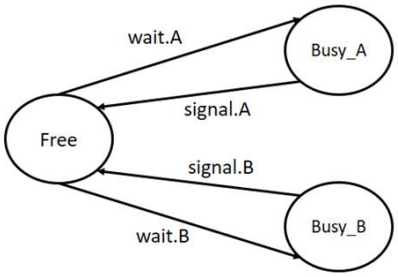
IV. Possible Approach (Mutex)

One of the possible solutions to fixing this concurrency issue, as described in Section III, is mutual exclusion. Mutual exclusion, or Mutex, is a process that serializes access.

For example, if DROPBOX = CLIENT || MUTEX, the FSM below describes the following.



A mutual exclusion (mutex) is a program that, when executed, averts concurrent access to a common resource. Only a single thread can own the mutex at a time, so when the program starts, a mutex is created. When a thread holds a resource, it must lock the mutex from any other threads to prevent concurrent access of the resource (to prevent starvation of data). On giving a signal to the resource, the thread unlocks the mutex and the resource is free for other threads to access [7].

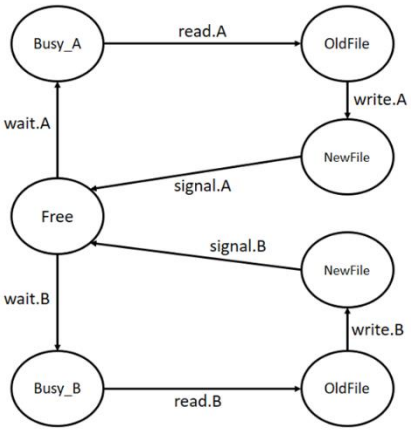


Once the first client accesses the file, the mutex will move from the state FREE to BUSY, using the action 'wait'. After the client is done with the file, it sends the 'signal' to unlock the mutex and return it back to FREE.

read.A-> write.A->read.B->write.B

This way, there is no interference since client A reads and then writes in sequence before moving onto the next client.

DROPBOX_SAFE = CLIENT_A || CLIENT_B || MUTEX



V. Discussion

A mutex is a locking mechanism used to coordinate access to a common resource. Only one task can attain the mutex at a time. It means there is ownership linked with mutex, and only the owner of the task can release the lock, as explained in Section IV.

A semaphore is more of a signaling mechanism ("I'm finished with the resource. You may move on"). For example, if you are listening to songs (this could be one task) on your mobile and at the same time, a friend calls you (a second task), it triggers an interrupt, which signals the call processing task to wake up [8].

In this paper, the problem was overcome using a Mutex semaphore. The Mutex would be locked only as far as the client is not finished with the resource. Once the client is finished, they will send a signal to the Mutex to unlock the resource for the next client to use.

One of the issues with Mutex is that if a thread acquires a lock and goes to sleep, the other thread, that is waiting in the queue, may never be able to move forward. It may lead to starvation. If Client A locks the file and then Client B requests to use the file, Client B would be added to a queue in order to use the file next. If Client A requests for the file again, they will be added to the queue and thus, creating a deadlock state in which no one can work on the file until the other is finished.

However, it is very easy to implement Mutex and since only one thread is working at any given time, the data remains consistent [9].

VI. Conclusion

This paper has introduced the concept of concurrency and Mutex, and provides the example of two users who are trying to edit a shared file on Dropbox concurrently. The problem of

interference due to concurrency was put forward (and also, graphically represented) and solved by using the Mutex semaphore.

But Dropbox isn't the only system that faces the concurrency issue. Through the analysis of this scenario, this solution can be implemented for potentially any system, and possibly modified, if necessary.

Therefore, it is very necessary to implement and experiment with mutual exclusion. Mutex may have its drawbacks, but it could prevent loss of data, especially if that data is important.

REFERENCES

[1] "BBC - WebWise - What is cloud computing?", *Bbc.co.uk*, 2012. [Online]. Available: http://www.bbc.co.uk/webwise/guides/what-is-cloud-computing. [Accessed: 29- Jan- 2018].

[2] "Lambda Days 2016", *Lambdadays.org*, 2018. [Online]. Available: http://www.lambdadays.org/lambdadays2016/john-hughes. [Accessed: 28- Jan- 2018].

[3] J. Armstrong, *Programming in Erlang*, 2nd ed. Prentice Hall, 2013.

[4] S. Vinoski, *Concurrency with Erlang*. IEEE Computer Society, 2007.

[5] "What is Concurrency? - Definition from Techopedia", *Techopedia.com*, 2018. [Online]. Available: https://www.techopedia.com/definition/25146/concurrency-programming. [Accessed: 17- Feb- 2018].

[6] R. Hall, *Concurrent Programming*. Dr. Richard S. Hall, 2001, p. 8.

[7] "What is Mutual Exclusion (Mutex)? - Definition from Techopedia", *Techopedia.com*, 2018. [Online]. Available: https://www.techopedia.com/definition/25629/mutual-ex`clusion-mutex. [Accessed: 30- Jan- 2018].

[8] V. →, "Mutex vs Semaphore - GeeksforGeeks", *GeeksforGeeks*, 2013. [Online]. Available: https://www.geeksforgeeks.org/mutex-vs-semaphore/. [Accessed: 14- Feb- 2018].

[9] "2 Advantages of mutex Easy to implement Mutexes are just simple locks that a", *Coursehero.com*, 2016. [Online]. Available: https://www.coursehero.com/file/p58v9u/2-Advantages-of-mutex-Easy-to-implement-Mutexes-are-just-simple-locks-that-a/. [Accessed: 17- Feb- 2018].