Concurrency issues within Cloud Systems

Sebastian Marian Zdroana
Middlesex University, London UK
ID No: M00495434
Email: sz207@live.mdx.ac.uk

Abstract – This paper is a documentation of our progress throughout our second year module CSD2600 Distributed Systems and Networking covering Concurrent and the issue it can possibly bring is not prevented within a multi-user system such as a cloud system (example of a cloud system: DropBox). This document goes into detail on the possible issue mentioned above and will provide methods to prevent the problem, such as the Mutex Semaphore.

Keywords – documentation, progress, concurrent, issue, possible, prevent, Mutex.

I. INTRODUCTION

What is Concurrency? Wikipedia describes it as "In computer science, **concurrency** is the property of program, algorithm, or problem decomposability into order-independent or partially-ordered components or units."[1] which is correct, however this will be described in more detail in the Background section of this paper.

This documentation presents the user with the concept of concurrent programming, what this is, the problem concurrency can bring with it and the possible solutions to prevent the problem which concurrency can bring with it. This problem will be described in text but also using diagrams such as FSM(Finite-State Machine) to help the reader easily understand the topic stated above (concurrency) and the problem it comes with.

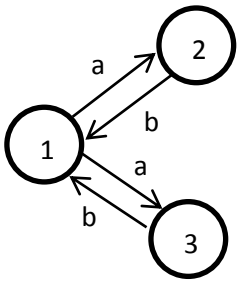
Firstly the reader will be presented with the concept of Concurrent Programming, then they will be enlightened on the issue it can bring, then a solution for the issue will be provided and an evaluation for the solution, and finally a conclusion for the whole topic.

II. BACKGROUND

Concurrency Programming, Wikipedia provides a more technical description of what this is but what does it really mean? Well imagine our day to day life, we are very concurrent about what we do; we wake up, we rant about what we have to do next; we wash our hands and face and brush out teeth; we get dressed and get ready to leave. However we never do more than one thing at once we do not start to get dressed and attempt to make a phone call at the same time however we can start to get dressed and then attempt to make a phone call however even so we can put on the wrong piece of clothing as we are talking on the phone and not fully paying attention to what we are doing or we can lose focus of what we are hearing on the phone while thinking of what to wear, don't get the image? What this means is that it's best to do one thing at once and not try to overload with too many things, that way everything being done can be done as it has to without derailing from the main expected outcome.

Concurrent programming follows the same concept, processes can start at different times and overlap each other, and this may not sound so bad for certain actions however what if a person is attempting to add £20 to their bank account and then attempt to add another £10 before the £20 is successfully added? Well the answer is the £20 can just be fully overlapped and lost, meaning that only £10 will be added to the account and the other £20 wont. This may not always happen, there can be situations where they just about manage to both update to the users account however there can be situations where they do overlap and money is lost which can cause a big issue to a company such as "Barclays" for example, which a lawsuit can possibly cripple them.

The scenario mentioned above is the exact same problem that can happen within a cloud system such as DropBox where a user can attempt to upload a file and then try to upload another file before the first file has been successfully uploaded, if the uploads overlap each other one of them can be lost.



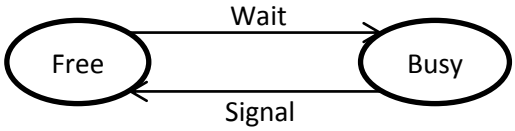
The above FSM shows two processes happening; it shows the state of the cycle, 1 to 2 to 1 to 3 and back to 1. However what's to say that the file has been uploaded at state 2 before state 3 starts? In theory they could both happen at the same time. The problem with this is the one

mentioned above; however there are solutions to this problem.

Another issue is when a user is attempting to make amendments to a file on their DropBox account that they may share with other people. If someone else attempts to open that file they will see the file changed this should not happen, a solution will be provided in the next section.

III. POSSIBLE APPROACHES

A possible solution to the first problem stated in section II of the paper is a "Mutex Semaphore" system. The Mutex Semaphore system can be described as a traffic light, the green is the start, the amber is the process going on and the red is the stop which means that the process has completed its cycle and another process can happen when the traffic light turns green. Basically the Mutex Semaphore would be called when a process begins to signal that it has started and that it is still processing. Now when another process attempts to start it will first check with the Mutex Semaphore that there are no occurring processes and if there is that it has completed its process before another one can be allowed to start. This Mutex Semaphore as already stated ensures that no two processes can happen at the same time, this prevents the overlapping overall but it makes the whole duration of the processes longer as the process has to wait for the previous process to finish its job before it can start its own.



The above FSM displays the Mutex Semaphore. It shows that only one process can happen at one time, for another process to start the current one must finish its course, after and only after it has completed its job can another process start. This shows how the overlapping can be combated.

A solution for the other issue addressed, the issue where a user can see a version of a document another user is working on is to not allow the user to see that version at all, but allow the user to see the original version, the version that was saved before the other user started working on it. This means that until the document is saved the user which is not working on it cannot see the changed that have been made by the user working on it until the document is saved. This takes the changed but unsaved data and puts it in a temporary log or file which when the file is saved is then updated to the server/database in place of the original. As long as the user who is updating the document does not save the document no other user attempting to view that document cannot see any change. This is so that the user who is not changing the data within the document cannot attempt to make a change while the person who first starting changing the document and in this way data cannot be lost.

IV. DISCUSSION

As an evaluation to the solutions mentioned in section III the Mutex Semaphore is a good solution as it excludes the possibility of overlapping processes as it loses data but it comes at the cost of causing the processes to take longer than they could if they were both happening at the same time, however this is not really a downside as personally I would rather know that all my files will be uploaded then knowing there's less than a 100% chance of them all being uploaded, or downloaded.

For the second solution my personal opinion is that a user shouldn't be allowed to view the changes made to a file until that file has been saved anyway as when someone is working on something you have no way of knowing if they are done with the changes they are making or knowing that they are making changes, as if you see a document you have no way of knowing that the document is being worked on as you are looking at it unless you refresh the document and see changes. I personally see the solution as common sense, only when the document has been saved should changes be made to the original document on the server/database.

V. CONCLUSION

In my conclusion I personally see these issues as something that happens when people are impatient. Why do five things at once with no assurance that they will all be done correctly? It's better to take things slowly and to think ahead of the possible outcomes, good or bad, and try to avoid the bad ones. With that being said in no way am I implying that this is not a serious issue within concurrency which can cause a lot of problems within a system but I am saying that the solutions provided are valid ones as they remove the possibility of overlapping processes and loss of data.

As future work I would much like to research more deeply into the issue as what I know what I have covered has not been described in as much detail as it could have been. I would like to explore the issue within other programming languages and not just Erlang, as concurrency is not just something that comes specifically with Erlang but in any other language that allows concurrent programming, processes happening at different points of time but never at the exact time.

VI. REFERENCES

[1] – https://en.wikipedia.org/wiki/Concurrency_(computer_science)
[2] – https://www.techopedia.com/definition/27385/concurrency-databases