Asynchronous Systems - Assignment 3

Muthukumar Suresh

1 Overview

I will be testing the performance of 3 TLA specifications of distributed Lamports' mutual exclusion Algorithms :

1. Module Timeclocks {filename: timeclocks & MC_timeclockcorrect} specified in `http://www.podc.org/podc2000/lamport.html`
2. MutexLamport(will be called mutex_merz in this doc) {filename: mutex_merz & MC_mutex_merz }from the Merz mail chain
3. LamportsMutex(will be called lamutex_merz in this doc), version 2 {filename: lamutex_merz & MC_lamutex_merz }from Merz mail chain (in Pluscal)

1.1 System Specs

1. RAM : 12.0 GB (51 % of the RAM was given to TLA+ which is around 6GB)
2. processor : intel i7, 2.30GHz, 4 core
3. Number of Worker Threads : 8
4. TLC Options : Model-checking Mode with depth of 100

2 Question 1

2.1 Size

Timeclocks is the smallest of the three specs. This is mainly due to the reason that Timeclocks was written in TLA instead of being converted from Pluscal like the other 2 algorithms. For the 2 Pluscal algorithms, I will not compare the size of the TLA spec since that is not the part that was written by Merz but instead I will compare the Pluscal specs.
lamutex_merz is the cleaner and smaller of the two specifications made by Merz.

2.2 ease of understanding

Timeclocks is obviously the easiest to understand of the 3 but that is because it is also the most simplest as it does not deal with how, for example, requests are added to the queue, how communication channels are designed, etc.
I find the lamutex_merz to be the best that manages to take all aspects like communication channel and critical section access at the same time keeping the Pluscal code easy to understand. Also, compared to mutex_merz, it has clear comments that explain the reasons for different parts of the spec. The use of macros for tasks like inserting, removing requests make the algorithm easy to understand.

2.3 Closeness to lamports Mutex Algorithm

We will compare the algorithms to the one described in Time, Clocks and the ordering of Events in a Distributed System

2.3.1 Clock Implementation

All three use a similar implementation of the logical clock and ordering (< operator is exactly same oh how they are implented and ties are broken). In timeclocks, each clock is updated as and when needed (it follows a literal transformation of the paper where between events when the clock is incremented to a number greater than present value instead of just saying 1). The Merz papers take a much more logical way of incrementing clock by just incrementing it by 1 when needed.

2.3.2 mutual Exclusion Algorithm and datastructures

timeclocks uses the simplest algorithm to that as the one specified in the paper but takes some shortcuts. While the paper only mentions a receiver queue. In order to implement some of the properties of the algorithm, extra datastructures like lastTSent and lastTRecvd are used to check message ordering but doing that makes the algorithm different from what lamport specified.

Merz specifications are much closer to what lamport specified in his paper. They use no extra datastructures than those that is specified in the paper. That is the main reason that the Merz code looks so long, because lets say when we need to check if a process can enter critical section, the code explicitely checks the 2 conditions specified by lamport :
1. oldest request 2. every other process request is higher LC time.

The lamutex_merz module goes even further and separates the communication framework and other tasks. This means that the spec can test asynchronous messages and other details that cant be tested in the other 2 algorithms.

2.4 correctness properties for each spec

2.4.1 Timeclock

Correctness properties tested:

1. Mutual Exclusion { file:MC_timeclockcorrect}− For timeclock, MutualExclusion<invariant> is satisfied.
2. Starvation Freedom < Strong Fairness>{file : MC_timeclockcorrect} - Fairness < property> is satisfied
3. Liveness { file : timeclocks} - is satisfied but it requires that you add AlwaysReleases[file : timeclocks] so that this condition is satisfied as shown in lamport slides.

2.4.2 MC_mutex_merz

Correctness properties tested:

1. Mutual Exclusion { file:mutex_merz}- For timeclock, MutualExclusion <invariant> is satisfied.
2. fairness < Strong Fairness> {file : mutex_merz} - Fairness <property> is satisfied
3. Liveness { file : mutex_merz} - is not satisfied but it seems to be odd why that is the case. In terms of the liveness expression, it is the same as that of timeclocks but I have not been able to reason out why.
4. Termination - also, fails but that might be because we are limiting the clock values.

2.4.3 MC_lamutex_merz

Correctness properties tested:

1. Mutual Exclusion { file:lamutex_merz}- For timeclock, MutualExclusion(invariant) is satisfied.
2. fairness (Strong Fairness) {file : lamutex_merz} - Fairness (property) is satisfied
3. Liveness { file : lamutex_merz} - is satisfied

3 question 2

3.1 Setting up and Running

Instead of talking about difficulty in setting up and running the specs, I will talk about what all needs to be done for testing using the TLC checker.
For making these algorithms work, while the TLA spec gives us the send, recieve and tick specifications. We need to find out how to make liveness work. To make it work, we need to make sure that AlwaysReleases is added to the spec. Otherwise liveness test for all the 3 specs will fail.

Also, We need to specify the constants (reference : lamport slides) for the processes and the maximum clock value. It took a bit of work to find out how to initialize the constants for timeclock and processes as the code is not that readily available but once this is done is trivial.
Note: the merz algorithm is in pluscal, so it requires that we convert that into TLA spec first using TLA+.
Finally, another important piece of code we need to add is the contraint for the clock [ref : MC_timeclockcorrect.tla] which makes sure that no clock value goes above the max_clock value. Since we are limiting the clock value, this is important. We may need to override some definitions but otherwise setting up is trivial. Note : one tricky thing was figuring out the value of any (which means none of the processes) in mutex_merz which comes from the translatlation from postcal to TLA but any value when set to 0 means that it hits none of the processes so the spec becomes correct.

3.2 Performance Results

Table 1: Timeclock Checker Performance

Procs	Max_Clock	Time taken (s)	States Found
2	5	2	35863
2	7	10	347580
2	9	61	2157110
2	10	142	4804214
3	2	1	6852
3	3	16	301626
3	4	342	5606881
4	2	90	1017859
4	3	> 3600	19507k

Table 2: lamutex_merz Checker Performance

Procs	Max_Clock	Time taken (s)	States Found
2	5	3	23888
2	7	5	67446
2	9	7	128362
2	10	9	164543
3	2	3	21646
3	3	35	578165
3	4	500	7763351
4	2	60	871209
4	3	> 3600	175359k

The values I chose for procs and Max_clock are the same as those that are there in the lamport slides. That gave me a useful criteria for comparison. I am not really looking at the time taken as that is dependent on the system.
In terms of nodes expanded, there is still some difference between the values that lamport slides give for timeclocks,

Table 3: mutex_merz Checker Performance

Procs	Max_Clock	Time taken (s)	States Found
2	5	1	3709
2	7	1	3709
2	9	1	4999
2	10	1	4999
3	2	80	243145
3	3	120	578165
3	4	120	578165
4	2	>3600	>170939833

either that is because of some setting difference or different properties being tested, so the spec formula might have been different.
In terms of performance, we can see that Timeclock is the most inefficient of them in terms of the model checker. That surprised me since I thought that if the spec is simpler then it should expand the least number of states but it makes sense after looking at the results that a more general spec will lead to more possible states.

For the merz algorithms, lamutex_merz seems to perform much worse than mutex_merz but that might be because of the overhead that lamutex has to accomodate for a separate communication channel. i.e., for n procs in lamutex, it creates $2 * n$ procs { n communication processes} which might contribute to the extra states.

But I am still not sure about the mutex_merz algorithm, I have noted in section 2.4.2 that the liveness and termination were failing and we notice here again how the performance of mutex_merz seems to be weird, example: 2 procs and 9 max clock has exactly the same states expanded as 2,10 and for 2 procs it is extremely fast. On the other hand, on 4 processes, it is extremely slow. I have not been able to reason out why this is happening, maybe it is just the thing about TLA : P