Report

Design

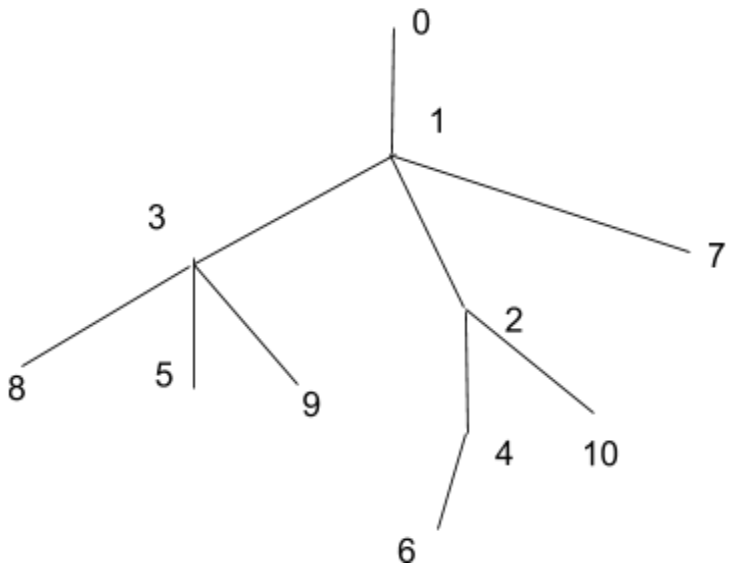
- OpenMPI was used to simulate different nodes
- Node with ID = 0 was treated as the coordinator node which send out snapshot requests and collect reports
- Each node(other than the coordinator) have 2 threads running in parallel to simulate receiving and sending of messages
- Application messages in Lai-Yang were sent to all neighbours of the process node of the application graph rather then only to its neighbours in the spanning tree
- Snapshots were sent to the coordinator along a spanning tree i.e, no out of band communication was used to send the snapshots to the coordinator
- Snapshots were sent directly to the coordinator instead of broadcasting in Chandy-lamport, this was done by building a dynamic spanning tree from the graph
- A **spanning tree** was assumed to be given in the input for **Lai-Yang** apart from the application graph
- **Sequence numbers** was used in **Lai-Yang** which is necessary in Lai-Yang algorithm to take multiple subsequent snapshots
- The following was treated as control messages: Termination, Snapshot, Marker(Chandy-lamport), the first red message(Lai-Yang)
- The rest of the design was made in compliance with the details of the algorithm mentioned in the textbook
- Details of how to run the code and how to pass the input can be found in the Readme

Details about sequence number in Lai-Yang:
https://www.cs.vu.nl/~tcs/da/daslides.pdf

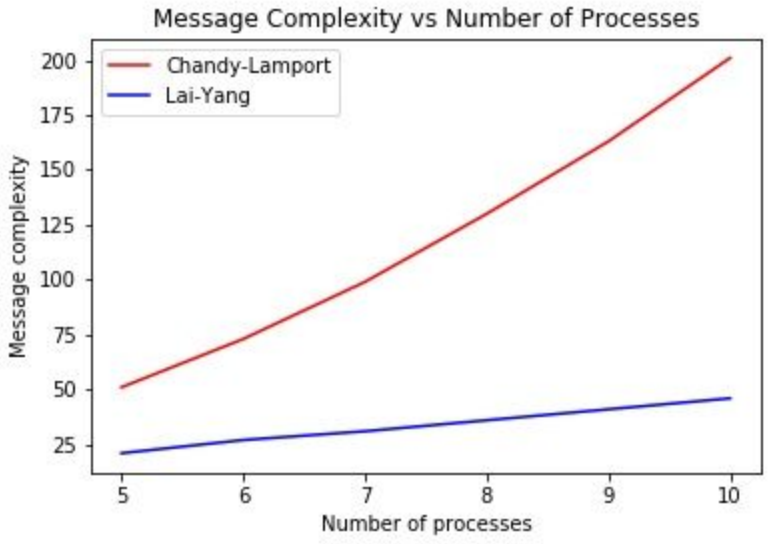
Graphs and Analysis

A fully connected graph was used for every run.

For lai-yang the following spanning tree was used over the fully connected graph.
(remove the id from the graph in descending order to get the subgraphs)



Node 0 was used as the coordinator
n= Number of processes, A=100, T=10, lamda=2 was used



It can be observed that Chandy-lamport's and Lai-Yang's message complexity forms almost a straight line with a higher slope for Chandy-Lamport's algorithm.
This is because Chandy-Lamport uses a broadcast scheme to distribute the marker messages to record its snapshot whereas Lai-Yang's algorithm works over a spanning tree.

Anomalies observed

We do not get a perfect straight line for Chandy lamport due to the following reason:
The snapshot messages were sent over a tree in chandy-lamport which was dynamically built in every run of the snapshot collection whereas in Lai-Yang, a spanning tree is assumed to be given in the input and this causes the deviation.

Errors encountered

Sometimes(very rarely), there is a seg_fault which occurs when working with higher values of n.

It is because of an openMpi issue rather than a bug in the code

Following is the snapshot taken while debugging the seg_fault.

```
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from ./lai...(no debugging symbols found)...done.
(gdb) r
Starting program: /home/abhi/Desktop/Sem7/Distributed Computing/Asn2/lai
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".
[New Thread 0x7ffff4d66700 (LWP 18667)]
[New Thread 0x7fffefff700 (LWP 18668)]
[New Thread 0x7fffd8c9f700 (LWP 18674)]
[New Thread 0x7fffd3fff700 (LWP 18687)]
[Thread 0x7fffd8c9f700 (LWP 18674) exited]
[Thread 0x7fffd3fff700 (LWP 18687) exited]

Thread 1 "lai" received signal SIGSEGV, Segmentation fault.
0x00007ffff645e3fb in ?? () from /usr/lib/x86_64-linux-gnu/libopen-pal.so.20
(gdb) bt
#0  0x00007ffff645e3fb in ?? ()
   from /usr/lib/x86_64-linux-gnu/libopen-pal.so.20
#1  0x00007fffe58b3391 in ?? ()
   from /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_btl_vader.so
#2  0x00007ffff6483229 in mca_base_component_close ()
   from /usr/lib/x86_64-linux-gnu/libopen-pal.so.20
#3  0x00007ffff64832b5 in mca_base_components_close ()
   from /usr/lib/x86_64-linux-gnu/libopen-pal.so.20
#4  0x00007ffff649f803 in ?? ()
   from /usr/lib/x86_64-linux-gnu/libopen-pal.so.20
#5  0x00007ffff648e1c1 in mca_base_framework_close ()
   from /usr/lib/x86_64-linux-gnu/libopen-pal.so.20
#6  0x00007ffff648e1c1 in mca_base_framework_close ()
   from /usr/lib/x86_64-linux-gnu/libopen-pal.so.20
#7  0x00007ffff79112d3 in ompi_mpi_finalize ()
   from /usr/lib/x86_64-linux-gnu/libmpi.so.20
#8  0x0000555555563208 in process(int, char**) ()
#9  0x0000555555563562 in main ()
(gdb) Quit
(gdb) Quit
```