Project-cs15btech11037_es15btech11002

by Abhinav Gupta

Submission date: 18-Nov-2018 10:06PM (UTC+0530)
Submission ID: 1041216210
File name: Project-es15btech11002-cs15btech11037.pdf (480.32K)
Word count: 1087
Character count: 5379

Report

Algorithms

We implement 2 algorithms related to optimizing the number of messages exchanged in the mutual exclusion problem

Singhal's Dynamic Information structure

This algorithm works on the idea that if a process does not enter the critical section, then other processes will eventually stop asking its permission to do so themselves. This is done via a initial initialization with respect to the process Ids of the various nodes as follows:
A process asks for the permission of every process with process Id lesser than its own.
Every node maintains 2 data structures to achieve the result stated above as follows: Request Set, Inform Set. The request set tracks which nodes permission to enter the CS has to be obtained from, while the inform set tracks which nodes need to be notified when this node exists the CS. The way in which the above said goal is achieved by this algorithm is as follows:
Whenever you get a reply message from a node, we delete its process ID from the requesting set and if this node does not or very rarely executes CS, its ID won't appear in the request set of other sites.

Agarwal - El Abbadi Algorithm

This algorithm improves the performance of quorum based mutual exclusion by using tree based quorums.
We implemented tree structured quorums (full binary tree) by using a mapping from process numbers to parent or child relationship in the tree structure.
Tree Structured Quorums:
In tree structured quorums, for a given node it's quorum set is all the ancestors of it including itself.
The best part is each quorum size is O(logN), therefore requiring very few messages per CS and therefore indirectly response time is also less.
Therefore we see for any two quorums atl east root of the tree is common in the sets.
Even though the responsibility of a node in tree based quorums is skewed (near to the root, more responsibility). Due to which it might not work when we have very large number of nodes due to heavy traffic at low level nodes such as root.

In terms of algorithm, It is similar to maekawa's algorithm. The messages involved and the message handlers invoked on receiving messages are similar.

Application

- The application developed to test the 2 algorithms is based on ATM service
- We assume that there is a single bank account associated with n number of ATMs and that amount can be credited and debited from any of the ATMs anytime
- We have put constraints on crediting and debiting the amounts to make the simulation more real

Design

- OpenMPI was used to simulate different nodes
- Each node have 2 threads running in parallel to simulate receiving of messages and working of critical section
- The following was treated as control messages: Request, Reply messages
- Report messages were used to send the report containing the statistics of the algorithm
- All the processes send their statistics to the coordinator process after completing their critical section k times
- Termination messages were used for the programs to terminate gracefully
- We make sure that the ATM has the latest view of the balance before executing the critical section through the reply messages sent to it from other processes, since we know that one of them was the last process to have executed the critical section and thus must have the latest view of the balance
- The rest of the design was made in compliance with the details of the algorithm mentioned in the paper
- Details of how to run the code and how to pass the input can be found in the Readme

Graphs and Analysis

Graphs

A fully connected graph was used for every run.

Scenario 1

Every process requests the CS 10 times with the following initial parameters:
Initial balance in the bank = 1,00,000

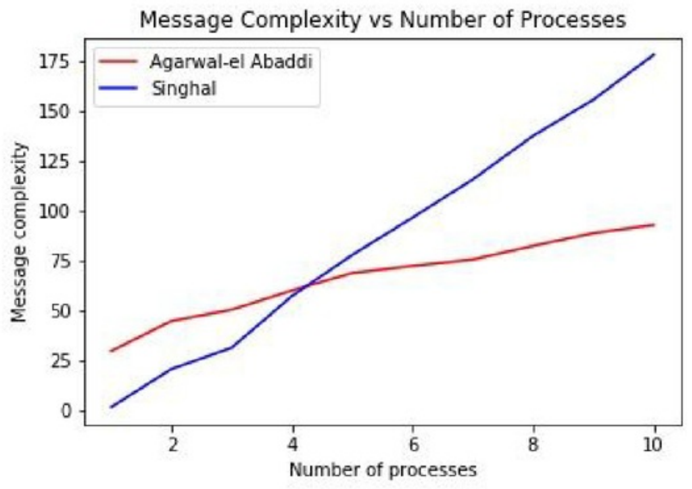


Figure 1

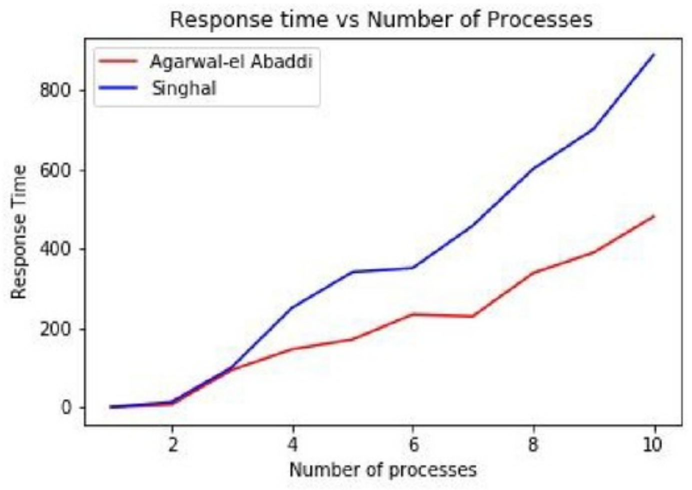


Figure 2

Scenario 2

Every process executes CS depending upon its process ID.
Here not every process requests the CS fixed number of times as the previous time.
By this, few processes become idle after certain point of time.
Every process requests CS 'max_transactions'/(procId+1) times. Where procId from 0 to n-1;

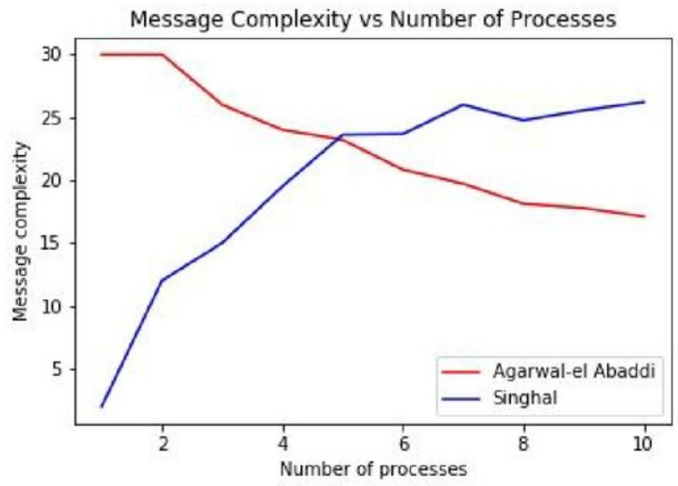


Figure 3

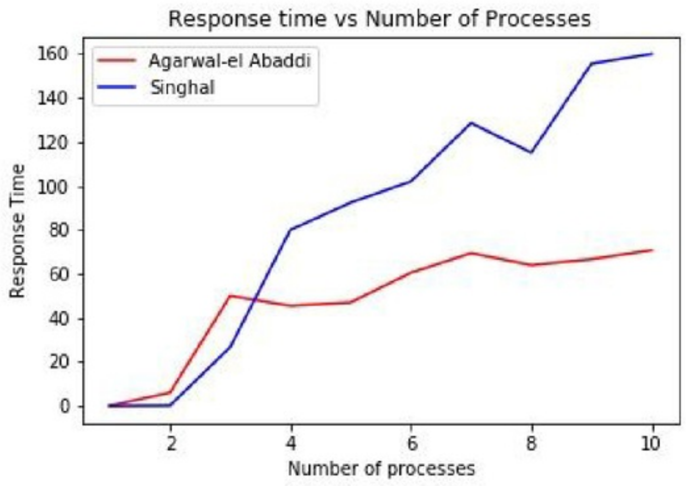


Figure 4

Analysis

Agarwal - el Abbadi Algorithm performing better in almost all cases: (Fig 1, 2, 3, 4)
It's easy to see that on average Agarwal El-Abbadi algorithm performs better due to it's very small quorum size (O(logN)), i.e., requires permission from just logN nodes, and hence less message complexity and less response time.
Where as for Singhal's algorithm, on average, it still needs linear number of request to send, and therefore requires more messages and hence higher response time.

- In Figure 1, 3 and 4, we can see that Firstly Singhal's algorithm performs better, but from N = 4 onwards Agrawal's performing better. It's because even though Asymptotic message complexity in Agrawal's algorithm is small (O(logN)) , the constant factor in Agrawal's algorithm is much higher than in Singhal's.   And therefore after a threshold N, this constant factor becomes negligible compared to N, and hence Agrawal's performs better. And the same reasoning applies for response time too.

- In Figure 3, we see Average message complexity of Agrawal's algorithm is decreasing with increasing N after some threshold. Because each process executes 'max_transactions'(10) / 'factor'(procId). There when we add a new process , in the average message complexity denominator changes from 'N' to 'N+1', but numerator no. of messages doesn't change significantly as we are having only 1 extra transaction. (for ex. N from 6 to 7 => increases 1 transaction [7th process executes 10/7 transactions = 1]

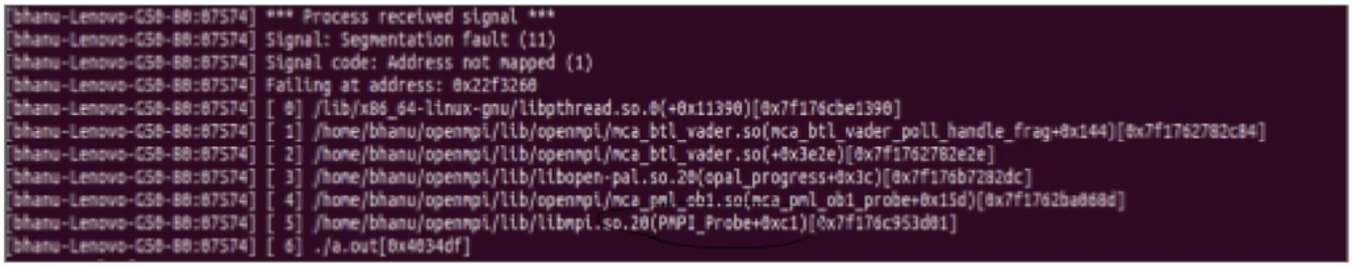
When does Singhal's algorithm work better:
Due to it's dynamic information structure, singhal's algorithm works best when only a few nodes require to execute CS in any instant. At any time, When multiple processes request for CS, then automatically request set and inform sets will be higher and thus increasing no of messages and therefore response time.

Errors encountered

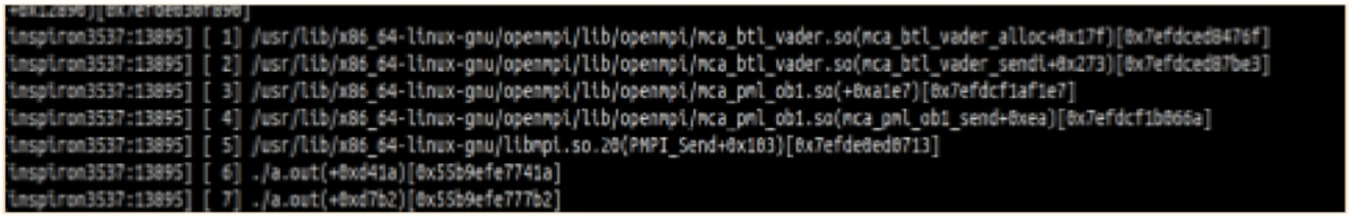
Sometimes(very rarely), there is a seg_fault which occurs when working with higher values of n.

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

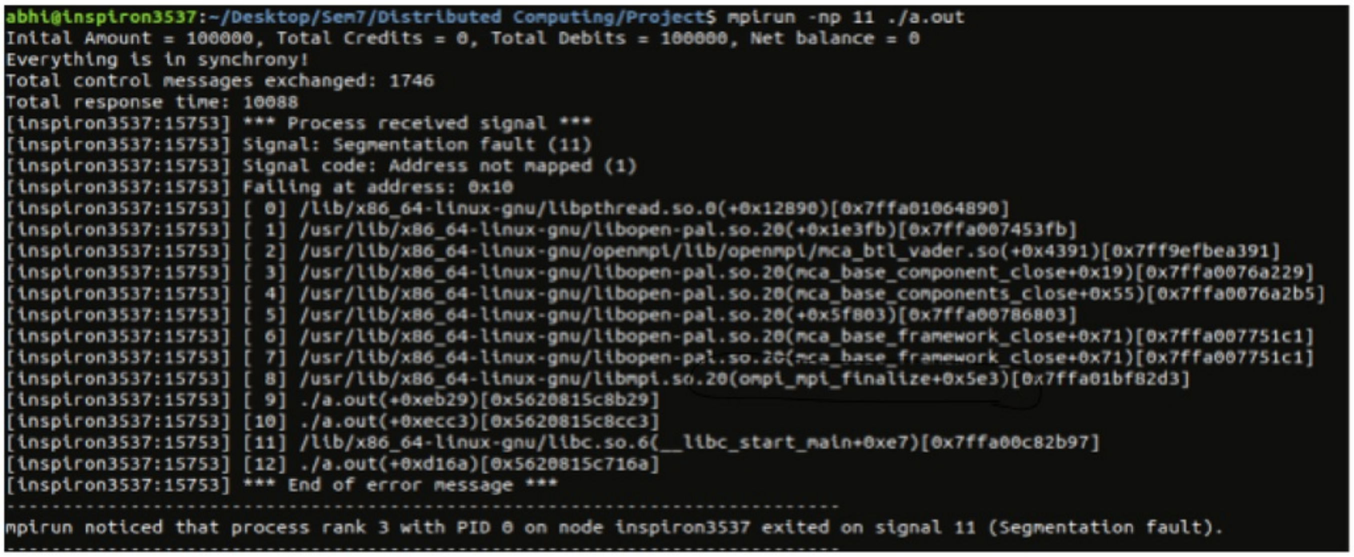
Following is the snapshot taken while debugging the seg_fault.



```
[bhanu-Lenovo-G50-80:07574] *** Process received signal ***
[bhanu-Lenovo-G50-80:07574] Signal: Segmentation fault (11)
[bhanu-Lenovo-G50-80:07574] Signal code: Address not mapped (1)
[bhanu-Lenovo-G50-80:07574] Failing at address: 0x22f3260
[bhanu-Lenovo-G50-80:07574] [ 0] /lib/x86_64-linux-gnu/libpthread.so.0(+0x11390)[0x7f176cbe1390]
[bhanu-Lenovo-G50-80:07574] [ 1] /home/bhanu/openmpi/lib/openmpi/mca_btl_vader.so(mca_btl_vader_poll_handle_frag+0x144)[0x7f1762782c84]
[bhanu-Lenovo-G50-80:07574] [ 2] /home/bhanu/openmpi/lib/openmpi/mca_btl_vader.so(+0x3e2e)[0x7f1762782e2e]
[bhanu-Lenovo-G50-80:07574] [ 3] /home/bhanu/openmpi/lib/libopen-pal.so.20(opal_progress+0x3c)[0x7f176b7282dc]
[bhanu-Lenovo-G50-80:07574] [ 4] /home/bhanu/openmpi/lib/openmpi/mca_pml_ob1.so(mca_pml_ob1_probe+0x15d)[0x7f1762ba068d]
[bhanu-Lenovo-G50-80:07574] [ 5] /home/bhanu/openmpi/lib/libmpi.so.20(PMPI_Probe+0xc1)[0x7f176c953d01]
[bhanu-Lenovo-G50-80:07574] [ 6] ./a.out[0x4034df]
```



```
+0x12890)[0x7efde030f890]
inspiron3537:13895] [ 1] /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_btl_vader.so(mca_btl_vader_alloc+0x17f)[0x7efdced8476f]
inspiron3537:13895] [ 2] /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_btl_vader.so(mca_btl_vader_sendi+0x273)[0x7efdced87be3]
inspiron3537:13895] [ 3] /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_pml_ob1.so(+0xa1e7)[0x7efdcf1af1e7]
inspiron3537:13895] [ 4] /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_pml_ob1.so(mca_pml_ob1_send+0xea)[0x7efdcf1b066a]
inspiron3537:13895] [ 5] /usr/lib/x86_64-linux-gnu/libmpi.so.20(PMPI_Send+0x103)[0x7efde0ed0713]
inspiron3537:13895] [ 6] ./a.out(+0xd41a)[0x55b9efe7741a]
inspiron3537:13895] [ 7] ./a.out(+0xd7b2)[0x55b9efe777b2]
```



```
abhi@inspiron3537:~/Desktop/Sem7/Distributed Computing/Project$ mpirun -np 11 ./a.out
Inital Amount = 100000, Total Credits = 0, Total Debits = 100000, Net balance = 0
Everything is in synchrony!
Total control messages exchanged: 1746
Total response time: 10088
[inspiron3537:15753] *** Process received signal ***
[inspiron3537:15753] Signal: Segmentation fault (11)
[inspiron3537:15753] Signal code: Address not mapped (1)
[inspiron3537:15753] Failing at address: 0x10
[inspiron3537:15753] [ 0] /lib/x86_64-linux-gnu/libpthread.so.0(+0x12890)[0x7ffa01064890]
[inspiron3537:15753] [ 1] /usr/lib/x86_64-linux-gnu/libopen-pal.so.20(+0x1e3fb)[0x7ffa007453fb]
[inspiron3537:15753] [ 2] /usr/lib/x86_64-linux-gnu/openmpi/lib/openmpi/mca_btl_vader.so(+0x4391)[0x7ff9efbea391]
[inspiron3537:15753] [ 3] /usr/lib/x86_64-linux-gnu/libopen-pal.so.20(mca_base_component_close+0x19)[0x7ffa0076a229]
[inspiron3537:15753] [ 4] /usr/lib/x86_64-linux-gnu/libopen-pal.so.20(mca_base_components_close+0x55)[0x7ffa0076a2b5]
[inspiron3537:15753] [ 5] /usr/lib/x86_64-linux-gnu/libopen-pal.so.20(+0x5f803)[0x7ffa00786803]
[inspiron3537:15753] [ 6] /usr/lib/x86_64-linux-gnu/libopen-pal.so.20(mca_base_framework_close+0x71)[0x7ffa007751c1]
[inspiron3537:15753] [ 7] /usr/lib/x86_64-linux-gnu/libopen-pal.so.20(mca_base_framework_close+0x71)[0x7ffa007751c1]
[inspiron3537:15753] [ 8] /usr/lib/x86_64-linux-gnu/libmpi.so.20(ompi_mpi_finalize+0x5e3)[0x7ffa01bf82d3]
[inspiron3537:15753] [ 9] ./a.out(+0xeb29)[0x5620815c8b29]
[inspiron3537:15753] [10] ./a.out(+0xecc3)[0x5620815c8cc3]
[inspiron3537:15753] [11] /lib/x86_64-linux-gnu/libc.so.6(__libc_start_main+0xe7)[0x7ffa00c82b97]
[inspiron3537:15753] [12] ./a.out(+0xd16a)[0x5620815c716a]
[inspiron3537:15753] *** End of error message ***
--------------------------------------------------------------------------
mpirun noticed that process rank 3 with PID 0 on node inspiron3537 exited on signal 11 (Segmentation fault).
--------------------------------------------------------------------------
```

Note above that the execution of the program completes gracefully as we get the output in the above image, but we get a Seg_fault in the MPI command, MPI_Finalize

Above shown are the errors encountered in OpenMPI in MPI_Finalize, MPI_Probe and MPI_Send .

Team Members:

Bhanu Prakash Thandu	(CS15BTECH11037)
Abhinav Gupta	(ES15BTECH11002)

Project-cs15btech11037_es15btech11002

ORIGINALITY REPORT

3%	0%	0%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Indian Institute of Technology Student Paper	3%

Exclude quotes	On	Exclude matches	Off
Exclude bibliography	On		