2022 Future Computing Summer Internship Project: (Mapping Deadlock to the SST Model)

Nicholas Schantz*, AnotherFirst AnotherLast†

July 15, 2022

Abstract

Deadlock is a common issue in distributed systems in which multiple nodes in a cycle cannot proceed with any action because they are waiting for each other to take action as well. SST will be used to model the problem and parameterized so that the problem can be simulated and metrics can be collected which identify if deadlock exist. Simulating the problem and determining if a system-level deadlock has occurred is possible, but progress needs to be made regarding if metrics exist for detecting if a component-level deadlock exist in a distributed system.

1 Project Description

The challenge addressed by this work is to map the distributed systems problem 'Deadlock' to the SST model. The problem is simulated to identify mathematical or logical conditions that cause the problem. This information is used to develop metrics to identify that the problem has occurred in generalized systems.

2 Motivation

Determining these metrics would allow for deadlock to be more easily identifiable in distributed systems then using algorithms to detect deadlock in the system. Furthermore, having metrics that identify the existence of the problem would help in creating better architecture to prevent the problem from occurring in distributed systems. Furthermore, the SST models written are resources that other users can use to learn and utilize SST's discrete event simulator.

3 Prior work

See https://doi.org/10.1145/357360.357365 for information on deadlock detection but unrelated to metric gathering. See https://doi.org/10.1145/3005745.3005760 for similar work in this area where metrics for causing deadlock were collected for a system utilizing Remote Direct Memory Access and Priority-based flow control.

4 How to do the thing

The software developed to respond to this challenge was run on one laptop. The software is available on (https://github.com/lpsmodsim/2022HPCSummer-Deadlock)

5 Progress

The model created for the deadlock problem was a network of n nodes in a ring topology passing messages along the ring in one direction.

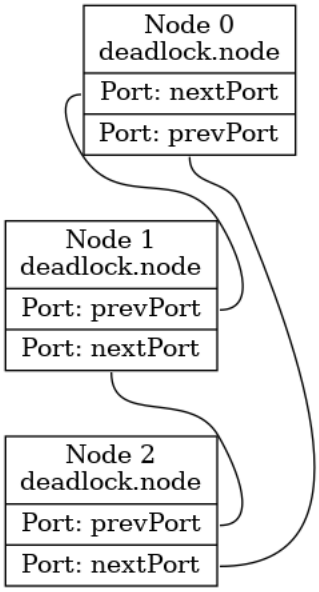

Figure 1: Example of a model with 3 nodes.

The model was built with the conditions for deadlock in mind:
- Mutual Exclusion - Each node has exclusive access to the next node's queue.
- Hold and Wait - Each node is making requests to send messages to the next node's finite queue.
- No Preemption - Nodes cannot force the next node to send messages out and free up its queue.
- Circular Wait - Nodes form a cyclic connection.

An issue with this model is that the conditions for deadlock occurrence are baked into the model so that deadlock will eventually occur. A better model would be to allow the conditions to be variable to get a more generalized system to collect metrics as to when deadlock exist in a distributed system. Furthermore, this model is one specific example of deadlock in which I can collect information on mathematical conditions for why deadlock is occurring in this model, but does not determine for all cases if deadlock has occurred. For example, in a communication model of deadlock where messages are passed along, node's send rates can cause deadlocks to occur if all node's queues are filled up before messages reach their destination. However, send rate could be irrelevant for a deadlock occurring between processes grabbing resources.

A second model was made which uses a logging node to keep track of information regarding all nodes in the ring. This gives more freedom on how we can you use the information shared between all nodes to form metrics to detect deadlock in the simulation. However, this model's fault is similar to the previous as it has the conditions for deadlock baked into the model.

6 Result

Results found so far is that system-level deadlock can be detected in a simulation.

In the first model of the simulation where nodes do not share information with each other, a single node can determine when it is in a blocked state. The node will detect that it has been in a idle state for a predetermined set of time and it has been making request for a node's queue for a large number of time. At this point the node will send out a probe along the ring to determine if other nodes are currently executing. If any nodes are executing, they will drop the probe, otherwise they will pass the probe along. If the sender node receives its own probe, it determines that all nodes are in a deadlocked state so system-deadlock has occurred.

In the second model of the simulation where nodes share information with a global logging node, edge probing is not required. The logging node can determine when all nodes are in an idle state and have made a large amount of request for a node's queue.

7 Future Work

Currently, more work needs to be performed on detecting component-level deadlock in a simulation.

References

[1] ADD BIB FILE

[2] ADD BIB FILE