2022 Future Computing Summer Internship Project: (Creating examples of livelock for the Structural Simulation Toolkit Discrete Event Simulator)

Melissa Jost*, AnotherFirst AnotherLast†

July 15, 2022

Abstract

The focus of this project was the current lack of examples looking into various distributed system design problems, and our ability to detect these issues. In order to address this, our goal within this repository was to take the issue of livelock, simulate it within SST, and come up with a set of metrics to help us analyze other systems to detect these issues. More specifically, we decided to simulate the dining philosopher's problem, and use different iterations of this problem to analyze if livelock had occurred. While coming up with the simulation for this problem was relatively straight-forward, there is still more work to be done in regards to solidifying and expanding on the metrics we chose to define this problem within distributed systems.

1 Project Description

The challenge that is being addressed by this work is that while we have countless examples of what issues existed within distributed systems, as well as various ways to avoid them , there is still a lack of work that exists with regards to quantifying metrics to detect these issues. In addition, by creating these simulations, we provide additional examples of livelock, which can be helpful to both those working to better understand these issues, as well as those trying to understand how to use SST as a whole.

2 Motivation

The users of this work include two main groups of people: those working on developing distributed systems, as well as those interested in learning how to use SST. For those working on developments for distributed systems, this is useful because if we are able to develop a truly useful set of metrics for detecting these various design problems, even if in simple simulations, it could potentially save a lot of time in finding them in real systems in the future. For those learning SST, these simulations help give more, simple examples of how to use the toolkit.

3 Prior work

See [**?**] for prior work in this area.

4 How to do the thing

For the simulation, we are able to run it with one laptop, with a total of five philosophers. This is how a majority of our analysis took place. However, by including additional links and components, we could easily scale this simulation to model a bigger system.

The software is available on (https://github.com/lpsmodsim/2022HPCSummer-Livelock)

In order to run it, you simply need to run the Makefile.

5 Models

Throughout the process of simulating the dining philosopher's problem, we were able to come up with a couple different instances of the problem, as well as note down certain parameters that seemed to make livelock more or less likely to occur. When looking into what components made the most sense to model, we had different iterations: one that focused on modeling philosopher components alongside a central dining table component, as well as philosophers alongside individual chopstick components. This allowed us to explore the ways in which different simulations would potentially affect the likelihood, as well as the ways in which livelock would occur. From here, we set a list of parameters in our simulations to see what would induce livelock. These parameters included:

- thinkingDuration: The amount of time a philosopher spends thinking after placing down his chopsticks, whether that was after he finished eating, or while trying to allow another philosopher to eat
- waitingDuration: The frequency at which a philosopher checks the state of his hands, and sees whether or not he needs to place down his chopsticks to allow another philosopher to eat
- eatingDuration: The amount of time a philosopher spends holding both chopsticks when eating
- randomseed: This is a seed for our random number generator that randomizes our first pass at grabbing chopsticks
- id: This id allows the chopsticks or the dining table to be able to identify the philosopher
- livelockCheck: This indicates how many cycles we want to run our simulation for before exiting
- windowSize: This tells us how often we should check our system for potential livelock conditions

The most important parameters in this list are the thinkingDuration and the waitingDuration, because these set up 2 different sets of clocks within our simulation that run on different cycle lengths. One cycle length is designated by thinkingDuration, and is where the philosopher sends requests to pick up chopsticks if necessary. The other cycle length is designated by waitingDuration, which is what allows us to induce livelock. Every time this clock cycle ticks, the philosopher checks the state of their hands, and if theyre only holding one chopstick, they put it back down to give someone else the opportunity to eat. This endless switching between looking for chopsticks in the thinking state, and placing them back down in the waiting state is what causes livelock to occur in the simulation.

In the model with a central dining table, we had 2 types of components: the dining philosophers and the dining table. We created 5 philosophers and 1 dining table in our python file, and gave each philosopher a link to the dining table. By doing so, we held all our chopsticks in one central location and made sure there were no duplicate variables. The philosophers would send message requests over their links if they needed to pick up a chopstick or put one back on the table. The dining table received these events, and would let each philosopher know whether or not those chopsticks were actually available or not. The information contained within each of those events is detailed in the ”requests.h” header file.

In our other model, we replaced the central dining table component with individual chopsticks. Instead of each philosopher only having one link to the table, both philosophers and chopsticks had 2 links to keep track of. Each philosopher had a link to both their left and right chopstick, and each chopstick had a link to their left and right philosopher. In a similar fashion to the model described above, philosophers can take possession or place down their chopsticks by sending requests over their links. The only difference here is that each chopstick only communicates with its adjacent philosophers as opposed to having one object managing the resources of the system.

One other model we had included, though not directly relevant to this issue of livelock, was a model that simulates deadlock. The only difference between livelock and deadlock in the dining philosophers problem is that the philosophers wont check to see if they need to place a chopstick down if theyre only being one. Within SST, this is what our waiting clock does. By removing the second clock, we force the philosophers into a deadlock, where each only holds one chopstick and no one ever eats.

When comparing both of these models, both of them produce very similar results, however, the model with individual chopsticks seemed to take longer to update their status. We can assume this would be due to the fact that there are double the amount of links in the chopstick model. We believe that the benefit that the chopstick model gives us is that from a logical point of view, it makes more sense to view the chopstick as its own component, as opposed to a dining table, which isn't something discussed as heavily in the definition of the dining philosophers problem. However, the benefit that the dining table model gives us is that it means we already have a central node in place that can double as a way to collect information from all the philosophers. In addition, since this model only requires each component to have one link, itd be easier to extend this model to have more philosophers or chopsticks.

Lastly, an important thing to note about these models is that they were designed specifically with the idea in mind that there would be the same number of chopsticks and philosophers. For example, the dining table keeps track of which philosophers would be allowed to access each chopstick. (ex. Chopstick L5R1 is only accessible as Philosopher 1s right chopstick, and Philosopher 5s left chopstick). The chopstick model also has a similar designation. This would be important to keep in mind if you wanted to alter the simulation to have an uneven number of chopsticks to philosophers, where certain components werent predetermined to only be accessible to a specific subset of other components. Generally speaking, the dining philosophers problem is designed to have an equal number of philosophers to chopsticks, but if you wanted to change the model in any way, this would be an important distinction.

6 Result

After we had established our models, we ran different sets of simulations on each set, observing when our philosophers actually experienced livelock. The main thing that affected whether or not we experienced livelock was the changes we made to the timing parameters of the simulation. Generally speaking, the more randomization you introduce into the timing, the less likely you are likely to experience livelock, with livelock being defined as the state in which none of the philosophers are ever able to enter their eating states. In addition, you generally want your waitingDuration to be at most, equal to the thinkingDuration if you want to experience livelock. In general, as long as each philosopher tries to grab the chopstick at the same time and puts them down at the same time, you are likely to enter a livelocked scenario.

In terms of the metrics we coined in order to detect livelock, our current findings is that they are heavily reliant on the specific problem at hand. For example, much of the work in this area tends to agree that livelock is dependent on an existence of some infinite loop that contains changing states, as well as the lack of progress in the system. In addition to this, many sources tend to agree that the definition of progress, as well as the definition of infinite is something that is left up to the programmer to decide, as it varies per problem. We can define those criteria in the dining philosopher's problem by looking towards how much time one spends switching between thinking and being hungry, and how often one eats.

Taking this information, we decided to define our metrics in terms of the progress we made in the system. We let our simulation run for an extended period of time defined by our parameters, and at that point check in on the status of each of our states: thinking, hungry, and eating. From here, we wait for another period of time that we designated as our "window". As this window closes, we once again take note of each of our states. Once we have this data, we can first note whether or not we allowed the philosopher to eat at all. This tells us whether or not we made progress in our system. After this, we can check the rate at which each philosopher switched from thinking to hungry. We do this by measuring in our window the amount of switches that happened, and generating a percentage of time spent in each state based on what we expect. In a livelock scenario, we would spend about half our time hungry, and half our time thinking, so we divide our window size by 2. This gives us a fraction of the actual number of switches into a certain state over the expected number of switches into a certain state. The higher each percentage is, the more likely it is to be livelocked, especially if there was no progress made in this window. In true livelock, our hungry boolean would show us that no one ate, and that each of our percentages show that the philosopher spent all their time in either the hungry or thinking state.

7 Future Work

Overall, the lack of concrete research in regards to both livelock as a whole, as well as the specifics of how to quantify livelock issues limited this project. Many papers had differing definitions on what qualified as livelock, and these various definitions were sometimes hard to translate into useful information for distributed systems, such as what would be considered an infinite loop, or what was meant by progress in the system. To further understand this problem, it would probably be most helpful to find more real examples of livelock (ex. not the hallway problem), so that we can come up with metrics that are applicable to a wider set of systems. While we can semi-confidently conclude that we have livelock in the simulations provided above, finding ways to generalize the process in which we found livelock would be useful (ex. a more general definition of progress, or clearer examples of states that don't only apply to dining philosophers).

References

[1] Donald E. Knuth (1986) *The TEX Book*, Addison-Wesley Professional.

[2] Leslie Lamport (1994) *LATEX: a document preparation system*, Addison Wesley, Massachusetts, 2nd ed.