Traversing async DAG

help

carlospereira1607 January 4, 2020, 1:46pm 1

Hello,

Currently I'm trying to get into Rust async/await and the Tokio crate. One of the things that I'm trying to do is implement an async version of a algorithm that I have that uses a DAG. This DAG that was created by me and under the hood there is a vec. I implemented the DAG as a vec because I want to reuse posititions from "deleted" nodes of the DAG (a node is just sofly deleted). I have 3 functions that update the graph: add (that might call update), update (which is recursive) and remove. When add is called, it checks the predecessors of that node in the graph (by trying to get the predecessors from the graph) and adds their indexes in the vec under hood to another vec that the struct node has.

What I did from my initial version without async/await was I transformed the functios add, update and remove as async methods and I wrapped the DAG on Tokio's Mutex. Whenever I want to mutate I call the .lock().await from the mutex, do the changes that I want and then I drop the variable so the lock is dropped.

The problem is traversing and mutating the graph. From my logs, if I add a node its predecessors (that already exists in the graph) are marked as not being there (even though the logs show that they are).

Another approach that I thought was having a Mutex at each node of the graph instead of wrapping the whole graph. This way, only one thread will have a lock at each node. Perhaps this second approach is better?

Regards,

raidwas January 4, 2020, 3:45pm 2

Without seeing the code the following is all guessing:
I assume the non-async version works?
Since the whole dag is inside a mutex there shouldn't be a race condition, so my guess would be that there is a problem in your data structure that gets triggered by a difference in call order from multiple threads compared to the single threaded version of your code.
Maybe try the await version with your single threaded algorithm?

About the other approach: without care that sounds very deadlock prone. You may want to read up on some of the required conditions for deadlocks. For example if you can make sure that all your resources (in this case nodes) have a unique id and every thread only allocates nodes in ascending order it would be impossible to create a deadlock.

Which approach is better (assuming better means faster, not simpler), it would depend on what you do with the data structure (for example how long does a thread lock the data structure (approach 1), how many mutually exclusive allocations would theire be etc (approach 2)).
But approach certainly sounds simpler than 2.

carlospereira1607 January 4, 2020, 5:50pm 3

Yes, the non-async version works (forgot to mention that). I did implement an async version that has the DAG struct wrapped in the Mutex and whenever its necessary to update it it calls .lock().await. Unfortunately, this version isn't that much better than my initial single threaded implementation. Since the functions that update the graph are still sequential and non concurrent, perhaps this makes sense.

In the end I just wanted to know if it would be possible to traverse a DAG asynchronously. The way I have the code it might not be possible to do it asynchronously, since I have to do a lot of comparisons between the node I'm currently visiting and its predecessors/successors.

alice January 4, 2020, 5:58pm 4

If you are not modifying it, you can share it with an `Arc` and just traverse it without any asynchronous functionality.

raidwas January 4, 2020, 5:58pm 5

If most of the time from the algorithm is spend in functions that work on `&must Dag` then it makes sense that the threaded version isn't much faster since most threads will just be waiting on the lock.

As for whether it is possible I would say yes, if it suits your needs is hard to say without knowing what you want to do with it.

carlospereira1607 January 5, 2020, 12:25pm 6

What I want to do is keep adding nodes to the DAG and when a node has all of its predecessors in the graph, I want to remove it (by marking it as deleted so the position can be later used), but before I remove a node, I want to remove all of its predecessors. When I add a node to the graph I already know which which nodes are its predecessors, but I don't know if they're already in the graph. If not, then the node will be in the graph until they are.

The idea of having the vec with the Mutexes doesn't sound very good to me, mainly because it will be shared by the threads and when a node were to be added to the graph, they would have to add it to this vec.

carlospereira1607 January 5, 2020, 12:26pm 7

I do need to be able to mutate the graph, so wrapping it in an Arc doesn't work... But thanks for the suggestion!

system Closed April 4, 2020, 12:26pm 8

This topic was automatically closed 90 days after the last reply. New replies are no longer allowed.

Related topics

Topic	Replies	Views	Activity
Safe-threading and async HashMap help	5	164	November 15, 2025
What makes async mutex more expensive than sync mutex? help	17	3059	January 5, 2024
How to access vector from multiple threads without mutex?	28	4151	July 28, 2023
Mutable struct fields with async/await help	20	5751	October 8, 2020
``HashMap::retain()`` and async/await help	4	1936	April 9, 2020