2022 CS6858: Distributed Trust

Module 2 Assignment 1

Due: 11 PM on Nov 5, 2022

The goal of this assignment is to test out the working of Hedera's Hashgraph and its consensus algorithm as described in this technical report. There are two parts. you must choose one part and implement it. You are expected to use google colab using your smail account.

Part 1: Creation of Hashgraph

This part is more open ended and requires you to write a tool that creates a variety of hashgraphs. Taking the number of member nodes n and the number of events m as initial parameters along with a type of hashgraph (as a string), you should create a hashgraph with n member nodes and of the stated type. For example, you may take as input `10 200 isolate` and create a hashgraph with n nodes and at least m events in which you isolate one member from others for a substantial period of time. Perhaps, another type may be `partition`, that partitions the network into two halves for substantial periods of time. Perhaps, you can create forks. And so on.

Your first goal should be to create a vanilla hashgraph where all member nodes behave reasonably nicely. Subsequently, build more types where you explore various ways in which hashgraphs can get messy due to either asynchrony or Byzantine behavior or both. You will be graded on the creativity you show in capturing various effects of asynchrony and Byzantine behavior.

You must also show a visual representation of the hashgraphs in DOT language. See below under section titled Common Visual Output.

Part 2: Implementation of the Consensus Algorithm

In this part, you will be required to read a hashgraph and order the events according to the hashgraph consensus mechanism.

Basic Output

You will be required to output the sequence of events output by the hashgraph consensus algorithm. Note that some events may not appear in the sequence. The output must be in the form of a file that has one event ID per line in sequence with the last line being "Done".

Common Hashgraph Format

Both parts deal with hashgraphs. The first part deals with the creation of hashgraphs while the second part deals with creating a total of events in a given hashgraph. So we now standardize the format used for representing a hashgraph.

The first line is a single positive integer n. Each subsequent line (barring the last line) is an event. For simplicity, we will not include transactions inside the events. Each event will comprise a node ID (integer between 1 and n representing the node that created the event), an event ID (a unique positive integer for each event), a parent event pointer and a self-parent event pointer followed by a time stamp (positive integer). Both pointers will be IDs of the respective parent/self-parent events. The last line will be a single word "Done". Below is an example. The first event in each node will have self-parent pointer as a null pointers and perhaps even the parent pointer may be null.

```
4
1   1231    NULL    NULL    2
3   7232    1231    Null    4
4   8234    7232    Null    5
2   7923    Null    Null    3
3   9812    7923    7232    5
2   9823    8234    7923    7
......
......
......
Done
```

Common Visual Output

Additionally, you are required to output the hashgraph in the form of a graph in the DOT language. For those who implement Part 2, you must also indicate the output sequence.

You can find details of the DOT language in https://graphviz.org/doc/info/lang.html. The language allows you to express graphs using various features such as labels on edges/vertices, colors and shapes for nodes, colors/thickness for edges, directed/undirected edges, and indicate subgraphs, clusters, etc. So your goal is to think of the cleanest way to express the hashgraph using the dot language so that it is easy to reason about it.

For example, you can place nodes in the same round in a separate subgraph. You can use directed edges to indicate the total order created by the hashgraph consensus algorithm. You can label the events with event IDs and the round numbers. Feel free to explore creative ways to express the hashgraph and the workings of the consensus algorithm through the DOT language.

IMPORTANT: Do not focus on large outputs. Instead, be creative in creating outputs that are small enough to easily comprehend the effects of the algorithm/asynchrony/Byzantine behavior while being large enough to express interesting behavior.

As in any creative process, you may want to start with some basic display of the hashgraph and then think of more clever ways to enhance it.

Policy on Collaboration and Sharing

Your code must be your own. You are not allowed to share code.

You are allowed and encouraged to share basic outputs before the deadline. For example, someone working on creating hashgraphs can share their initial/basic hashgraphs. This will be useful for those implementing the consensus algorithm. They can use your hashgraphs as test cases. And, vice versa as well. If you share your executable consensus algorithm, those students creating hashgraphs can see their effects on the consensus algorithm. You are encouraged to share these things via the online forum on moodle so that the whole class can benefit.

For creative work, it is perhaps in your interest not to share too much information (until after the deadline).