Homework

due October 3, 2024

Simulate gossiping in a distributed system.

You are to simulate the effect of gossiping with adjusted eagerness in a large distributed system of 2000 nodes. (You do not have to build a distributed system.)

Each node has a clock and a rate at which it makes contact. The standard and maximum rate is 1000. When a node is created, its clocks is set to a random value between 0 and 1000. Whenever the clock is larger or equal to the rate, then the node contacts an arbitrary other node and the clock is reset.

Upon contacting another node, the nodes compare their value. If the node contacts another node with the same value, then the rate is increased by 200, but never more than to 1000. If one of the nodes has a stale value, then both nodes “become eager” by setting their rate to 10.

We simulate by giving one node a new value and set its clock equal to the rate, so that it shares its news with a random node. We then display the result of five random simulations.

You can use the following code or one you write yourself in a language of your choice. In the latter case, you will probably have to create the graphics using a spreadsheet.

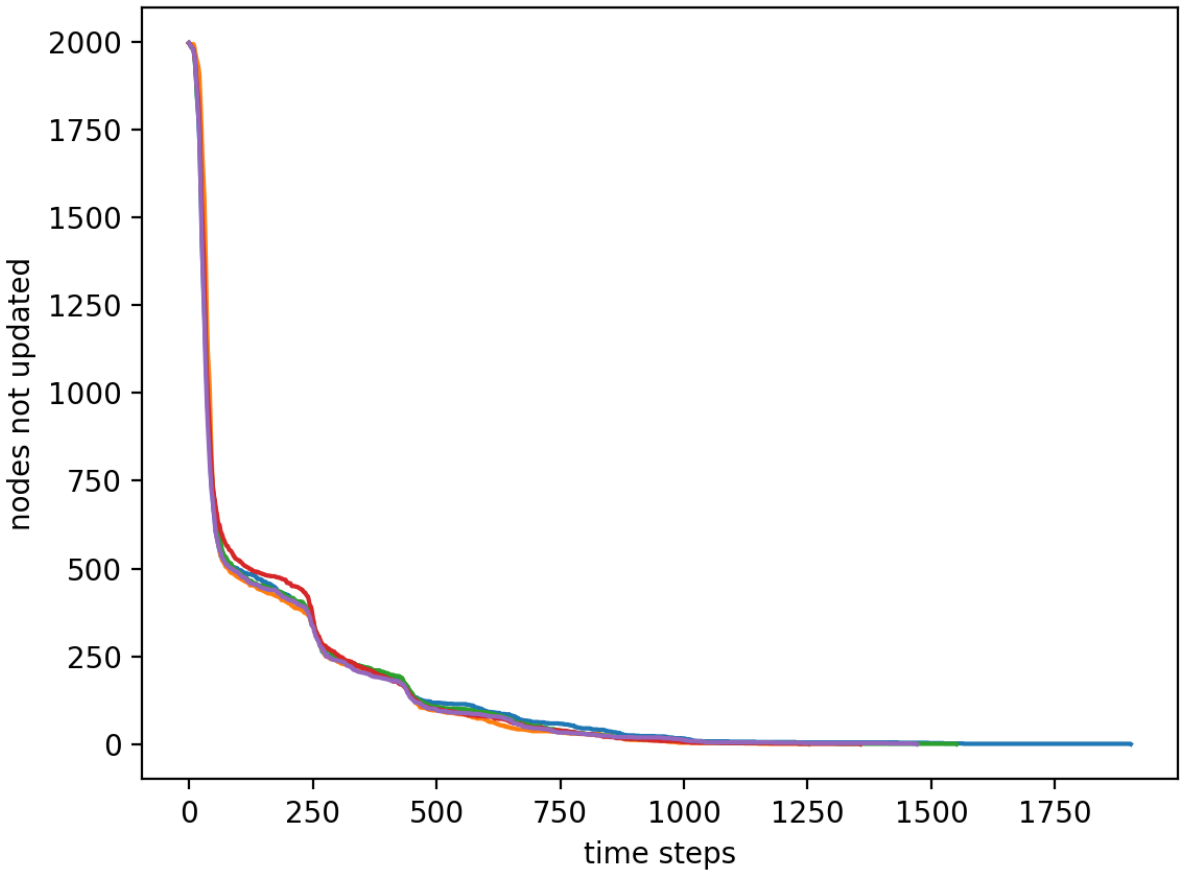
Task:

(1) We modeled a push-pull, where the contacting node can be updated by the contacted node. Change the code to a push model, where only the contacted node will change value and its behavior.

(2) Change the code to adjust so that initially, all nodes have a value 0. One node then gets an update to 1 and another node gets an update to 2. If two nodes are in contact, then the higher value wins. Plot the number of nodes with value 0, with value 1, and with value 2.

```
import random as rd
import matplotlib.pyplot as plt

class Node:
    next_id = 0
    def __init__(self):
        self.id = Node.next_id
        Node.next_id += 1
        self.rate = 1000
        self.clock = rd.randint(0, 1000)
        self.updated = False
    def __str__(self):
```

```
        return f'Node {self.id}, rate {self.rate}, state
{self.updated}'
    def send(self, other):
        if self.updated and not other.updated:
            other.updated = True
            other.rate = 10
            self.rate = 10
        elif self.updated and other.updated:
            self.rate = min(self.rate+200, 1000)
            other.rate = min(self.rate+200, 1000)
        elif not self.updated and other.updated:
            self.updated = True
            self.rate = 10
            other.rate = 10
        elif not self.updated and not other.updated:
            self.rate = min(self.rate+200, 1000)
            other.rate = min(self.rate+200, 1000)

def count(nodes):
    result = 0
    for n in nodes:
        if not n.updated:
            result += 1
    return result


def simulate():
    numbers = []
    nodes = [Node() for _ in range(2000)]
    nodes[0].updated = True
    nodes[0].clock = 10
    nodes[0].rate = 10
    while True:
        for n in nodes:
            if n.clock >= n.rate:
                n.clock = 0
                other_node = rd.choice(nodes)
                n.send(other_node)
            n.clock += 1
        c = count(nodes)
        numbers.append(c)
        if c == 0:
            return numbers


for _ in range(5):
    res = simulate()
    plt.plot(range(len(res)), res)
plt.xlabel('time steps')
plt.ylabel('nodes not updated')
plt.show()
```


2000
1750
1500
1250
1000
750
500
250
0
nodes not updated
0
250
500
750
1000
1250
1500
1750
time steps