FEBRUARY 26, 2021

REAL-TIME INTELLIGENT SYSTEMS

ASSIGNMENT 7

ARPIT PARIHAR
UCID - 12261599

Contents

Problem Statement

You are chosen to design an infrastructure to design a bank account where 20 people are going to share. This bank account will have an initial balance of 0. Folks can randomly make a deposit and withdraw money from this account. This balance should always be coherent with the transaction time. It means if a transaction T1, T2, T3 are created respectively in this order, the balance should reflect this same order. For instance, T1 add $10, T2, add $5, T3 remove $10 the balance should be sequentially $10, $15 and $5.

This structure will be free of failures. It means you will need to find a way to store the balance somewhere. If this location is removed, you should not lose the balance information. Every agent (people owning this bank account) will be able to support agent leaving this bank account or being added to this bank account. You will need to think about the scalability, the transaction ledger, the balance coherency, the failure in the system, the speed.

You will propose an infrastructure respecting the above requirements.

Solution Architecture Overview

A distributed hash table can be used to address the problem at hand, where each person sharing the bank account is a node, and the ledger is distributed across all nodes in a way such that no singular transaction is exclusively stored on a single node.

Following are the ways in which the proposed architecture addresses the requirements of the system:

Robustness to failure:

Redundancy is a key feature of the architecture, each node can store information about its own transactions, and transactions of up to **r** other nodes, where r could be any number depending on the budget of the client.

This assures that if 1 node fails, the data stored in it is also available elsewhere. Every time a node fails/quits or joins the network, an update function needs to be called to make sure that the ledger is redistributed, and redundancy is maintained.

The figure below illustrates a DHT with 7 nodes, where each node captures its own transactions and the transactions made by the previous node (r = 1). The color of the line in the ledger indicates the color of the node which made that transaction. Each transaction here is contained within 2 nodes, assuring that if a node fails, the data isn't lost.

If a new agent joins or if an agent leaves, the ledgers will be updated to make sure that redundancy is still maintained, and no node contains exclusive information.

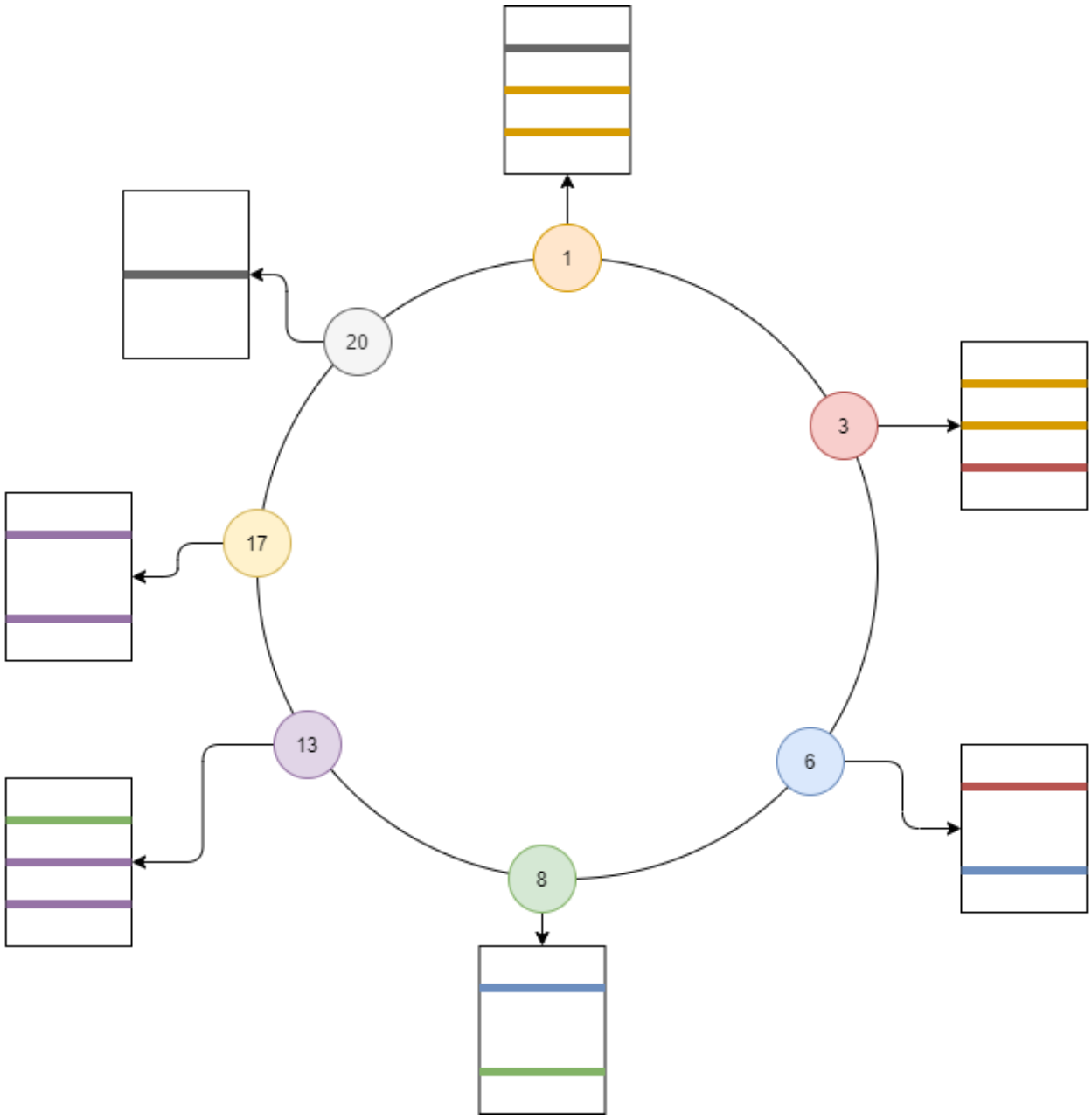

Figure 1

Scalability

The total number of nodes allowed in the network is dependent on the number of bits needed to create a node ID. As of now, the system only needs 20 members, but the system can be designed to handle 2^k members, where k is the number of bits. For k = 20, the system can be scaled up to 1,048,576 members.

This doesn't change anything in the design concepts of the system, as each time a node is added to an empty spot, all the transactions of r previous nodes will be assigned to this new node, and all the transactions made by this node will be stored in r subsequent nodes, making the system extremely scalable.

Balance Coherency

The order of transactions is challenging to manage as there's no global clock in distributed systems, but this problem can be solved by using Lamport's logical clocks.

The idea behind logical clocks is that instead of maintaining real-life time, each process (or node) maintains a logical sequence of events, which could be as simple as monotonically incrementing a counter by an arbitrary number. Whenever an event occurs, it is assigned a timestamp which is the current value of the counter.

The events (or transactions in this case) can happen within a node or can be sent/received to and from other nodes. Each received transaction needs a separate timestamp in the order of events of that node, which is calculated as ***max(clock of current node, timestamp of send event) + 1*** to maintain the order of events:

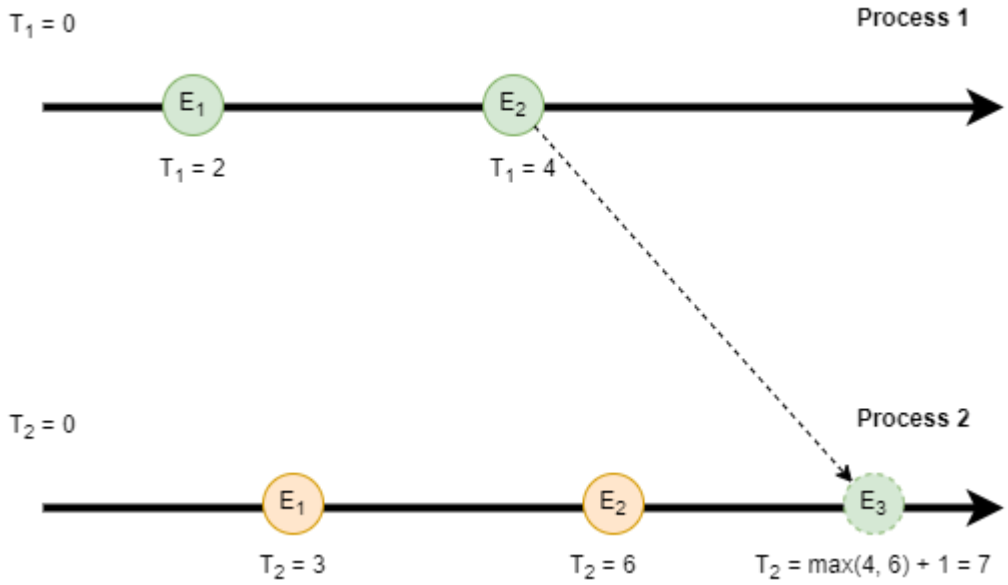

Figure 2

Speed

The problem with DHT is, to query the contents of a node, one would have to go through each node from the head of the chain to the target node in steps of 1, as each node points only to its following node. This, however, can be easily remedied by creating finger tables for each node, which is a table of pointers to k other nodes, where k = number of bits in the id.

Finger tables cut the querying time by an order of 2 at every step, making the time complexity O(logN). Following diagram illustrates the difference in speed when querying a certain node:

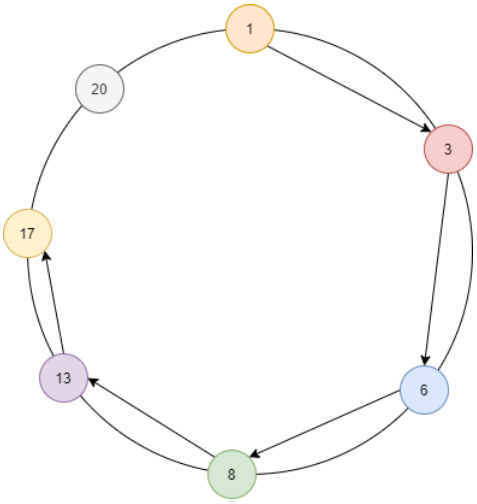

Figure 3 - Querying Node 17 without finger table - 5 hops

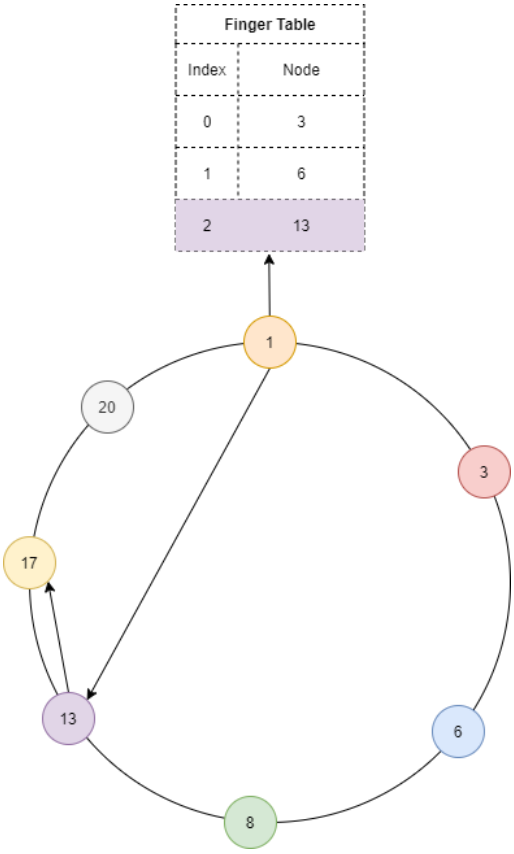

Figure 4 - Querying Node 17 with finger table - 2 hops