NoSQL Assignment-2(Part A)

Abhinav Kumar - IMT2022079
Siddhesh Deshpande - IMT2022080
Krish Patel - IMT2022097
Jinesh Pagaria - IMT2022044

February 28, 2025

Problem 1

1)

Yes, the M-Counter qualifies as a Conflict-Free Replicated Data Type (CRDT). It satisfies the 4 mentioned CRDT properties:
Associativity: For any three state-based objects x, y, and z:

$$\text{merge}(\text{merge}(x, y), z) = \text{merge}(x, \text{merge}(y, z))$$

Since the `merge` operation performs a pairwise maximum of the two arrays and the max function is associative, the result is the same regardless of the grouping of operations.

Commutativity: For any two state-based objects x and y:

$$\text{merge}(x, y) = \text{merge}(y, x)$$

The pairwise maximum operation is symmetric, so the result remains unchanged no matter the order of the inputs.

Idempotence: For any state-based object x:

$$\text{merge}(x, x) = x$$

Taking the maximum of an array element with itself leaves the array unchanged, ensuring that repeated merges with the same state don't alter the result.

Monotonicity (Increasing Updates): The `add`(x) method only increments values in the array and never decreases them. These kind of updates which always increase some element(s) in the array make sure that the new state will always dominate older in the pairwise maximum.

2) Table 1

Server	**State (`xs`)**	`query()`	**History**
a	$a0 : i = 0, n = 2, xs = [0, 0]$	0	$\{\}$
a	$a1 : i = 0, n = 2, xs = [1, 0]$	1	$\{0\}$
a	$a2 : i = 0, n = 2, xs = [1, 2]$	3	$\{0, 1\}$
b	$b0 : i = 1, n = 2, xs = [0, 0]$	0	$\{\}$
b	$b1 : i = 1, n = 2, xs = [0, 2]$	2	$\{1\}$
b	$b2 : i = 1, n = 2, xs = [0, 6]$	6	$\{1, 2\}$
b	$b3 : i = 1, n = 2, xs = [1, 6]$	7	$\{0, 1, 2\}$

Table 1: For refernce see state diagram

3)
CRDTs ensure eventual consistency and convergence, meaning that all replicas will eventually reflect the same state, though not necessarily immediately. Whether to use CRDTs or not depends on the kind of application being considered.

For example, consider a collaborative text editor where a group of people work on a shared document. Some users may be online while others may be offline. In this case, the users prefer an unobstructed ability to make edits, while consistency can be deferred to the merge time, which will eventually happen when offline users sync their operations. Here, correctness may sometimes be compromised (like possible duplicate edits), but users prioritize seamless control while editing. The document will become consistent 'eventually'.

On the other hand, take the case of banking applications. Transactions in this context require strict consistency because resolving conflicts later could impose a very high cost.

Problem 2

The Output of the mapper when it processes a document is of the format `(doc-ID,(index,word))` where key is doc-ID and value is `(index,word)`. The Reducer takes this as input and then emits key value pairs of the format `(index,(doc-ID,word)` .

So Basically on Seeing the output of the reducer it will appear as If each document is thought of as a different version or snapshot of the same document, with the document ID acting as the snapshot ID, then each key-value pair emitted by the reducer would indicate the index

position and the respective changes that occurred at that index across different timestamps.

The Time Taken for the mapreduce job to run on the larger dataset is 3182.682 seconds which is approximately 53 minutes .

The System Configuration of the System on which the mapreduce job was executed as follows:

- CPU:Intel i7(12th Gen)
- Cores: 9
- RAM : 9661 MB
- OS : Linux Mint

Problem 3

the input is of the format :

```
(index, (doc-ID, word))
```

which is read by the mapper and emits key value pair in this form :

```
(index, (timestamp, word))
```

Here document ID is converted to timestamp by converting it to long (UNIX timestamp i.e. no of seconds from JAN 1,1970).

The reducer receives the key value pair emitted by mapper after shuffling. So it has list of pair of timestamp and word (values) for a particular index (key). We then choose the the word which has the highest timestamp and emit that key value pair as output for that index (key).

But the length of final output may be larger than the the file with latest timestamp as any other file with older timestamp may have more words than the file with the latest timestamp. To fix this issue we must maintain a global latest timestamp variable at the reducer and emit key value pair if it is equal to the latest timestamp.

We have created a python script to check if the output of mapreduce exactly matches the file with latest timestamp and it does match .

We have submitted the code for both the programs where the output produced has discrepancy and the one producing correct output as well as their respective output files.

NOTE - The system configuration is same as problem 2.

The Runtime for Both the mapreduce jobs are as follows:

1. The Time taken for the mapreduce job executed by the file that produces output with discrepancy is 40.551 seconds.
2. The time taken for the mapreduce job executed by the file that produces correct output is 41.260 seconds.