Map Reduce

Table of Contents

1. Map Reduce for Word Count

```
Assume you are tasked to write a MapReduce-based application (a Map function and a
Reduce function) that analyzes the way words are used in documents. Each input
document corresponds to a separate Map task where Reduce function needs to know
the total number of times each word occurs in the total set of documents.

A database server is setup that has a table mapping words to counts. Map function
is programmed to connect to that database over the network and tell it to
increment the relevant count in the database for each word that Map sees in its
input. Reduce function is written to consult the database when it needs a word’s
total count. Counts are reset carefully in the database to zero before starting a
MapReduce job. A single MapReduce job is run at a time.

However, to your surprise, you find that sometimes the counts that the Reduces
read from the database are too high – higher than the total number of times a word
actually appears.
```

1.1 Describe what might have caused this issue.

```
Assuming only a single instance of the 'Map' is running at a time, that means
whenever the 'Map' finds the word, it triggers an 'IncDB(word)' i.e. Only one
increment at a time. If there's a network failure, the message will not be
recieved by the DB, which means no increment takes place. That means, as long as
there's only 1 'Map' task running, the DB is safe from incorrectly writing a value
greater than the actual value to the DB, because a single 'Map' cannot incorrectly
trigger an increment more than once, whenever it finds the matching word.

However, even if the issue is about reading 'higher than the total number of times
a word actually appears', with a single 'Map' task, if sent 'IncDB(word)' fails,
it means that the actual increment is not written to the DB i.e. Reduce reads
lower than the total numner of times a word actually appears.
```

msg	word	recieved	count
-	w	-	0 (init)
1, IncDB(w)	w	t	1
2, IncDB(w)	w	t	2
3, IncDB(w)	w	t	3
4, IncDB(w)	w	f	3
5, IncDB(w)	w	t	4

```
So, the issue has to occur due to running multiple 'Map' tasks even though it is
said to be 'single instance at a time'.
```

1.2 Suggest a possible solution to this problem. You need to specifically look in to the distributed system features.

```
The 'Map' task only triggers the 'IncDB(word)' function whenever the word is
found. Assuming there are 3 'Map' tasks running at them same time but with some
transmission delays in delivering the message to DB.
```

Map task	delay
M1	0.5 s
M2	0.0 s
M3	0.8 s

```
Now to fix the issue, we can have the 'IncDB(word)' to keep track of the last read
value in the DB i.e convert 'IncDB(word)' into 'IncDB(lastCount, word)'

This way, the increment can be prevented if the lastCount value is not greater
than or equal to the actual DB value i.e. prevents 'Map' tasks from writing higher
counts than the actual.

It's important to note that LOCK/UNLOCK alone cannot solve this issue i.e. This
would only prevent the multiple 'Map' tasks to access the specific record
simultaneuosly. It doesn't prevent the other 'Map' tasks to incorrectly increment
the value eventually.

Another way to solve this is to make sure the 'Master' kills any 'Workers' who run
the same 'Map' task i.e. The 'Master' pings the 'Workers' periodically and
whenever 'Master' finds a 'Worker' who runs the same 'Map' task, that 'Worker'
must be immediately killed. However, this still doesn't solve the issue for lost
messages that causes the 'Reducer' to read low count than the actual count.
```

1.3 Comment on the efficiency of the database that is utilized here and suggest a better approach to handle it.

```
Based on the problem scenario, we can assess which factors are likely available
and which are not available in the given database:

Available:
```

```
1. Concurrency: The database allows multiple Map tasks to access and update word
counts concurrently over the network.
2. Network Latency
3. Data Model: The database has a schema mapping words to counts

Not available:

1. Scalability: does not explicitly mention how to accommodate a large number of
concurrent requests.
2. Locking Mechanisms: no mention of specific locking mechanisms.
3. Indexing and Query Optimization: There's no indication of optimization
strategies for queries or indexing mechanisms within the database.
4. Fault Tolerance: There's an indication of handling failures by resetting counts
to zero before each MapReduce job, suggesting some level of fault tolerance.

So a better database with the following properties would be ideal to tackle the
efficiency drawback in the existing database.

A distributed key-value store with strong concurrency controls to manage
simultaneous updates from Map tasks. Implement effective locking to maintain
consistency and prevent overcounting. Making sure the database is horizontally
scalable to handle a high volume of concurrent requests and data. Ensure fault
tolerance with features like automatic failover and data replication. Optimize
indexing and query performance for efficient word count retrieval. Implement
monitoring and management tools for proactive maintenance. This database design
will efficiently support word count processing in the MapReduce job while ensuring
scalability, fault tolerance, and data consistency.
```