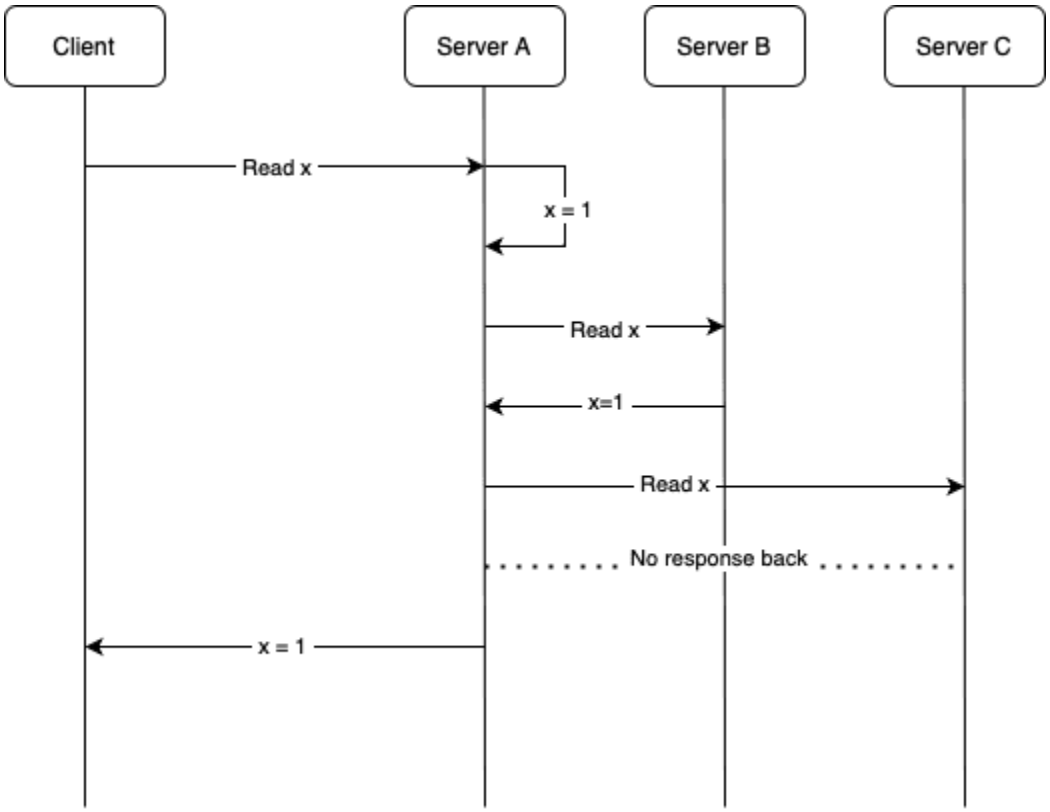
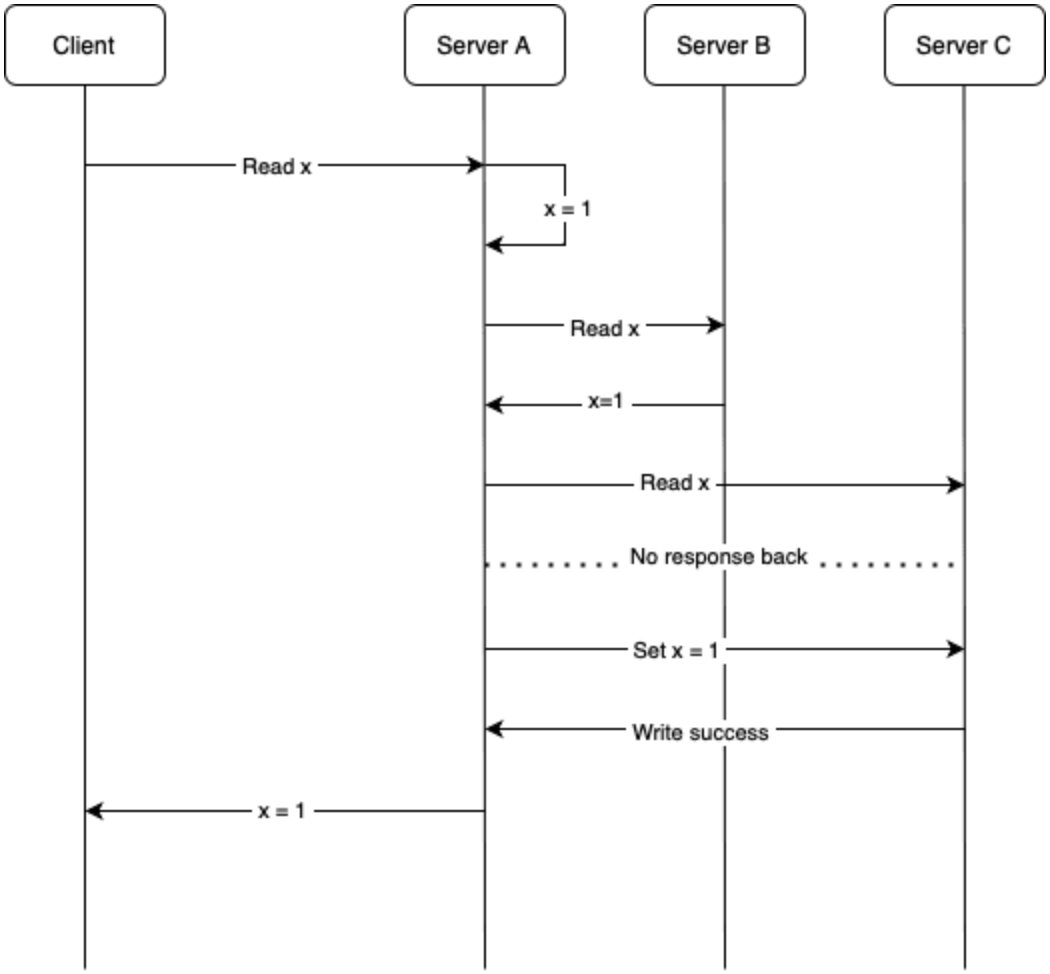
In a recent workshop on Distributed Systems, I was introduced to the concept of the majority quorum. I found it quite interesting, particularly the part surrounding the read repair pattern. This post will give an overview regarding the same, along with comparisons of two of their variants.

What is a majority quorum? In a distributed system, where there are replicated servers present, an action is made by ensuring that the majority of the servers would have succeeded in that action. For instance, consider the diagram below, with servers A, B and C. If a user requests a read operation on server A, then only once at least 2 of the 3 servers returns the response of the operation, will the result be returned back to the user, thereby concluding the read.



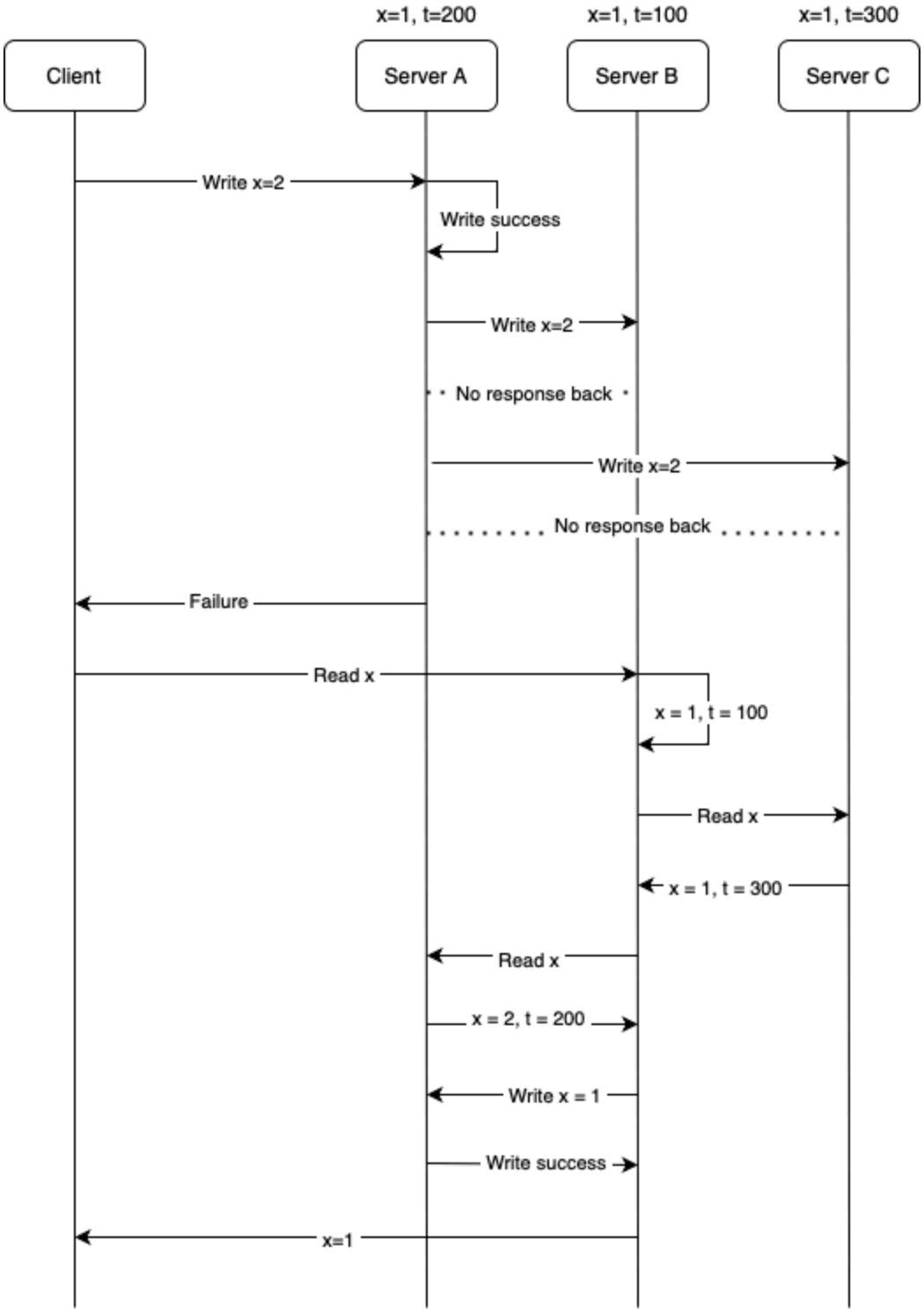

Read repair is an enhancement on the above. It is best explained using the above example. For the same servers A, B and C, on achieving majority quorum, if we write to the remaining servers the result of the majority, or in other words, repair the remaining servers, this pattern is achieved. Here we ensure that all the servers would be consistent for that particular data after the read operation. The repairing action can happen synchronously or asynchronously.

Synchronous read repair involves repairing the minority servers after achieving quorum and then sending the success response back to the user. This is best depicted on the diagram below. Here, the response times are significantly affected, since we have to not only wait for quorum, but also for the repairs to complete. Additionally, if the repair operation fails, then the entire read operation would return a failure response, even though we were able to extract the response of the same operation.

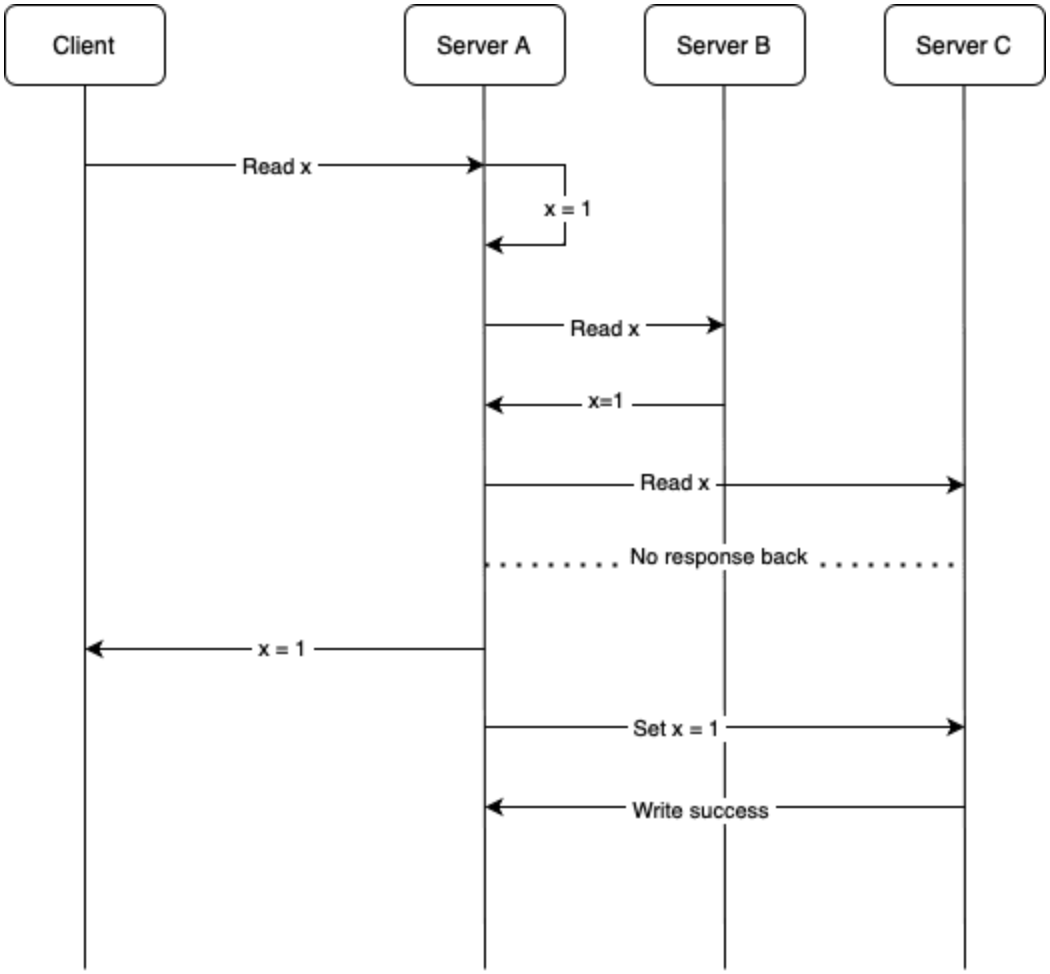



A straightforward enhancement that can be made to protect the integrity of data could be checking the timestamps before changing any data, so that the request with the latest timestamp would be executed.

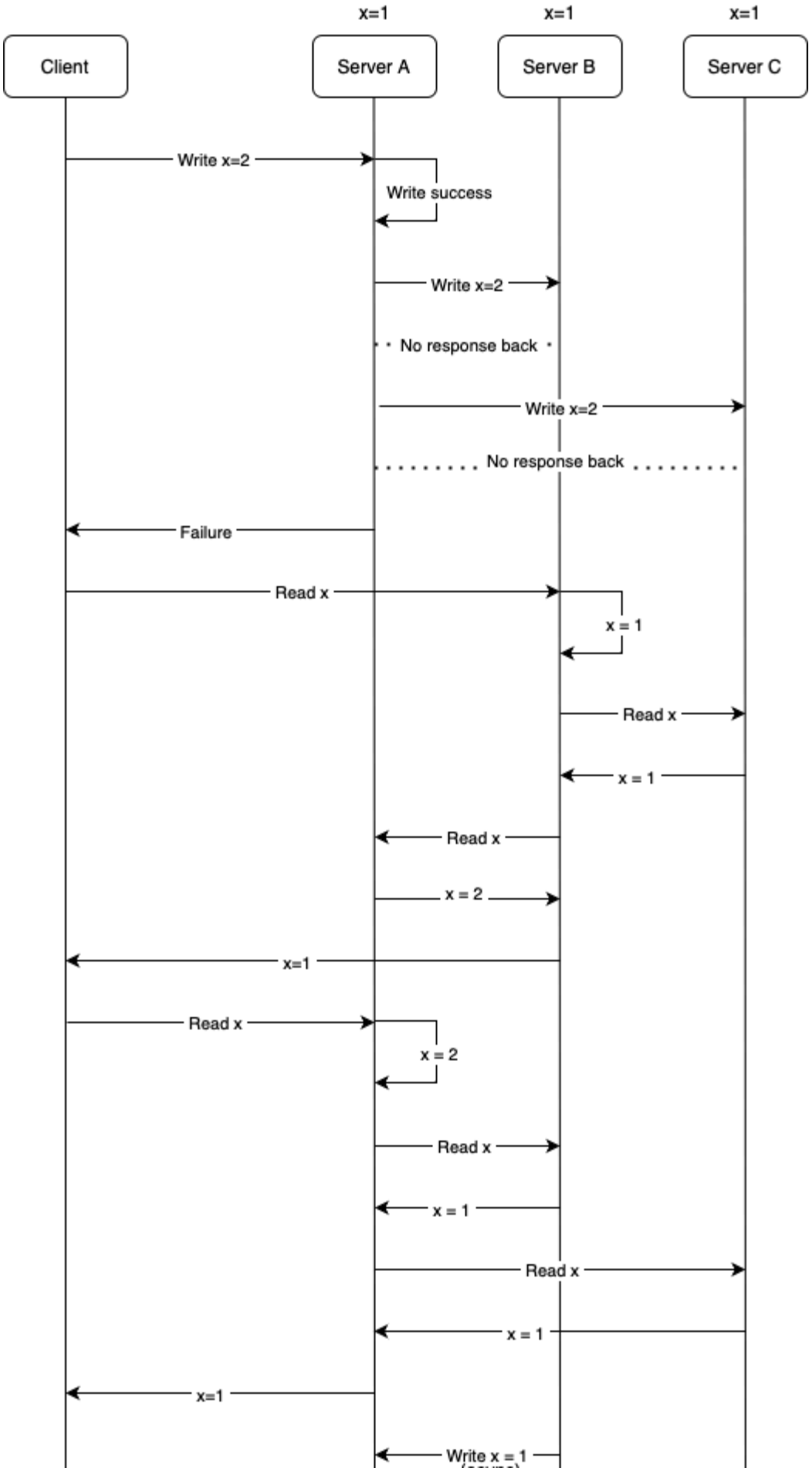
However this introduces another problem called clock skews which are a major inhibiting factor. The combination of incomplete write operations along with clock skews can cause an issue in consistency of the data. This can be explained by the following example: Consider 3 server replicas A, B and C, whose clock timestamps are 200, 100 and 300 respectively(overly simplified) and let x = 1 be stored in all three servers. Now A gets a write request for x = 2, but after writing to A, the remaining writes are unsuccessful. This is an incomplete write operation. Following this not much later, if B gets a read operation, then x would be updated back to 1 in sync read repair.




Asynchronous read repair, involves performing a read operation, and then repairing the other servers at a later time. This has an obvious benefit of ensuring a shorter response time due to some work being delegated to a later stage, but it also has some disadvantages.




Assume an incomplete write operation has happened. This means that only a minority of the servers have the new value. If a subsequent read operation happens, the async repair of the incorrect nodes would happen much later, during which many other read operations would happen, triggering their own repairs for the same nodes. To illustrate this, consider 3 server replicas A, B and C, and let x = 1 be stored in all three servers. Now A gets a write request for x = 2, but after writing to A, the remaining writes are unsuccessful. This is an incomplete write operation. Following this not much later, if B gets a read operation, then x = 1 would be returned according to the majority quorum. But before the async repair of A, if another read request is sent, A would still return x = 2. Now all the read requests before repair would get this response from A, triggering their own repairs to happen later.


x=1
x=1
x=1
Client
Server A
Server B
Server C
Write x=2
Write success
Write x=2
No response back
Write x=2
No response back
Failure
Read x
x = 1
Read x
x = 1
Read x
x = 2
x=1
Read x
x = 2
Read x
x = 1
Read x
x = 1
x=1
Write x = 1

Hence both types of read repairs have their advantages and disadvantages. There are also fixes present for some of the above mentioned disadvantages. However it is not in the scope of this post. To learn more about this, refer to the book Patterns of Distributed Systems by Unmesh Joshi.

To summarize, this is one of the many methods to ensure consistency in a distributed system. It is fairly straightforward and reliable, since it prioritizes the value of the majority, but has its drawbacks. Read repairs provide a venue to fix these inconsistencies, but do so at the expense of latency.