15. Vector Clocks

Let's learn about vector clocks and their usage.

We'll cover the following

Background#

When a distributed system allows concurrent writes, it can result in multiple versions of an object. Different replicas of an object can end up with different versions of the data. Let’s understand this with an example.

On a single machine, all we need to know about is the absolute or **wall clock** time: suppose we perform a write to key `k` with timestamp `t1`, and then perform another write to `k` with timestamp `t2`. Since `t2 > t1`, the second write must have been newer than the first write, and therefore, the database can safely overwrite the original value.

In a distributed system, this assumption does not hold true. The problem is **clock skew** – different clocks tend to run at different rates, so we cannot assume that time `t` on node `a` happened before time `t + 1` on node `b`. The most practical techniques that help with synchronizing clocks, like NTP, still do not guarantee that every clock in a distributed system is synchronized at all times. So, without special hardware like GPS

units and atomic clocks, just using wall clock timestamps is not enough.

So how can we reconcile and capture causality between different versions of the same object?

Definition#

Use Vector clocks to keep track of value history and reconcile divergent histories at read time.

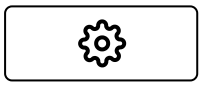
Solution#

A vector clock is effectively a `(node, counter)` pair. One vector clock is associated with every version of every object. If the counters on the first object's clock are less-than-or-equal to all of the nodes in the second clock, then the first is an ancestor of the second and can be forgotten. Otherwise, the two changes are considered to be in conflict and require reconciliation. Such conflicts are resolved at read-time, and if the system is not able to reconcile an object's state from its vector clocks, it sends it to the client application for reconciliation (since clients have more semantic information on the object and may be able to reconcile it). Resolving conflicts is similar to how Git works. If Git can merge different versions into one, merging is done automatically. If not, the client (i.e., the developer) has to reconcile conflicts manually.

To see how Dynamo handles conflicting data, take a look at Vector Clocks and Conflicting Data (https://www.educative.io/courses/grokking-adv-system-design-intvw/JE7p471DQ3l)

Examples#

To reconcile concurrent updates on an object Amazon's **Dynamo** uses Vector Clocks.



← Back

Next →

14. Checksum

16. CAP Theorem

Mark as Completed

Report an Issue