**What is the article about?**

This article about how use a service that exposes the existence of vector clocks in its API. It provides useful information with examples about how vclocks works in detail. Also, it provides solutions as “to prune”, when vclocks began to grow fast.

**What do vclocks guarantee?**

These clocks expand on Scalar time to facilitate a causally consistent view of the distributed system, they can guarantee the detection whether a contributed event has caused another event in the distributed system. It essentially captures all the causal relationships. This algorithm helps us label every process with a vector(a list of integers) with an integer for each local clock of every process within the system.

**How do vclocks solve conflicts?**

The options like last-write-win are always proposed in addition to vector clocks or as their replacement. But it's so easy to lose data using last-write-wins conflict resolution in a key/value system. Vector clocks solve this problem by allowing the database to push conflict resolution back out to the client. The simple rule is: assign each of your actors an ID, then make sure you include that ID and the last vector clock you saw for a given value whenever to store a modification. Vector clocks are good at helping clients with simple merges, but it's important to understand that vector clocks only tell you that a conflict occurred, and not how to resolve it;

**Explain the concept of pruning and reasons.**

If we use client identifiers, we’re back in the situation where vector clocks will grow and grow as more clients use a system over time. The solution most people end up with is to “prune” their vector clocks as they grow.

This is done by adding a timestamp to each field, and updating it to the current local time whenever that field is incremented. This timestamp is never used for vclock comparison — that is purely a matter of logical time — but is only for pruning purposes.

This way, when a given vclock gets too big, you can remove fields, starting at the one that was updated longest ago, until you hit a size/age threshold that makes sense for your application.

**What are the other timekeeping tools besides vclocks?**

Some noteworthy logical clock algorithms are:

* Lamport timestamps, which are monotonically increasing software counters.
* Vector clocks, that allow for partial ordering of events in a distributed system.
* Version vectors, order replicas, according to updates, in an optimistic replicated system.
* Matrix clocks, an extension of vector clocks that also contains information about other processes' views of the system.