High Availability under Eventual Consistency (incomplete)

Contents

Latency magnitudes

- λ, up to 50 ms (local region)
- Λ, between 100ms and 300ms (inter-continental)

In **planet wide geo-replication:**

- Consensus/paxos $[\Lambda, 2\Lambda]$ (with no divergence)
- Primary-Backup $[\lambda, \Lambda]$ There's always a replica that is faster (the primary) and the others just follow. (asynchronous/lazy)
- Multi-Master λ (allowing divergence), only local interactions.

Eventually consistency

In an ideal world, all the systems would have linearizability: when an update is made, all observers would see that update. But this is not possible, due to the time of propagation: all replicas must wait the update to be done in order to proceed.

In worldwide scale, it’s necessary to have trade-offs between the consistency and availability.

Eventual consistency is a **weak consistency.** After an update is made, if the no new updates are made, all read operations will read the same value. The DNS does it.

CAP

The shared-data systems must provide:

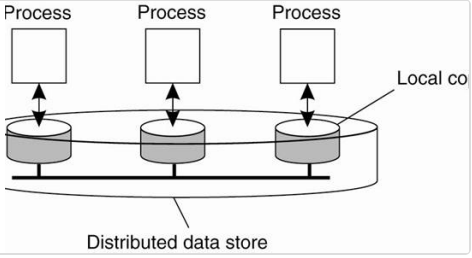
- Data consistency
- Availability
- Tolerance to network partition

Session guarantees

Chapter 7

Introduction Reasons for Replication To keep replicas consistent, we generally need to ensure that all conflicting operations are done in the the same order everywhere Guaranteeing global ordering on

https://csis.pace.edu/~marchese/CS865/Lectures/Chap7/Chapter7fin.htm



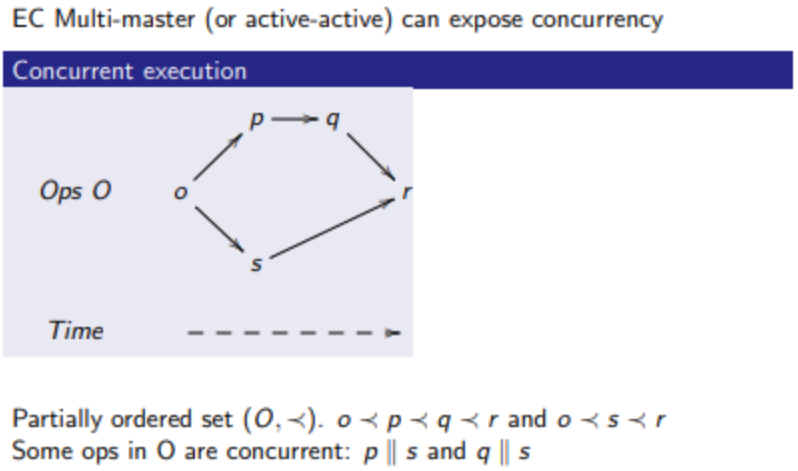
- **Read your writes**: read operations reflect previous writes
- **Monotonic reads**: successive reads reflect a non-decreasing set of writes. A read will always return the same value or the most recent one. It’ll never read a previous version.
- **Writes follow Reads:** writes are propagated after reads on which they depend. In other words, any successive write operation to a sequence of write and read, will overwrite the same value seem in the read operation or a more recent one. $W_a \rightarrow$

$R_a \rightarrow W_b$. Example: See reactions to posted articles only if you have the original posting (first we write, then we read the operation to write a reaction).

- **Monotonic Writes:** writes are propagated after writes that logically preced them. In other words.

EC multi-master

Is a method of data-replication, that allows data to be stored in a group of computers. This can expose concurrency.



Conflict-free replicated data types (CRDTs)

It's conflict-free because we can always merge the data without conflicts. That's the value of this technique.

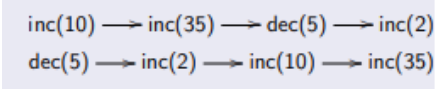
The point is that some datatypes are commutative, such as:

- PN-counter: inc(dec(c)) = dec(inc(c))

The thing is that, for any pair of sets, we can join the data without conflicts.

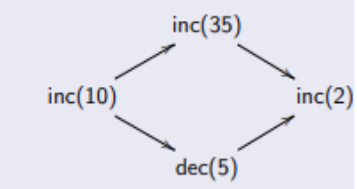
Principle of permutation equivalence

Imagine that we have a set of operations. These operations, if follows the commutative property, the

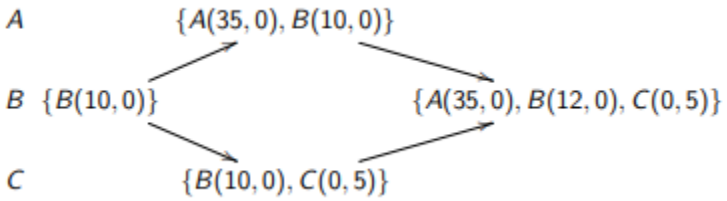


set of the linear applies returns always the same result:

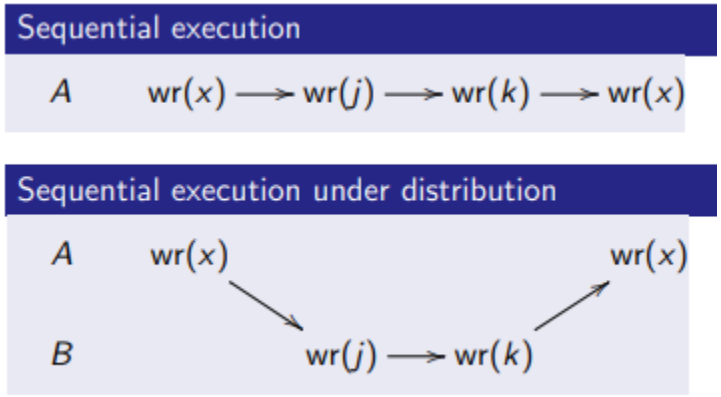
For this reason they can applies in parallel without problems:



A PN counter, for example can be seen be merged without problems:



Register (dynamo cart)



Preservation of sequential semantics

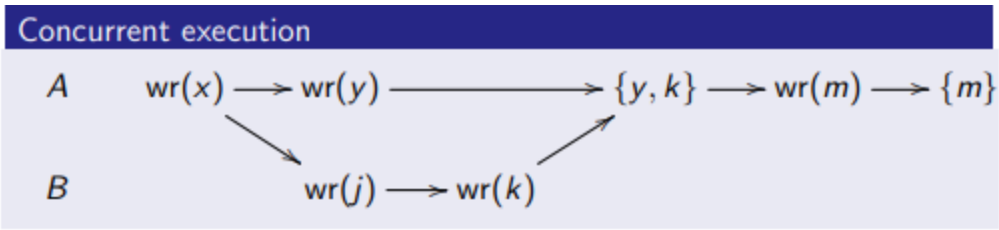
Concurrent semantics should preserve the sequential semantics. If the program was executed in parallel, then we should be able to also be executed in sequence. This also ensures correct sequential execution under distribution.

Multi-value register.

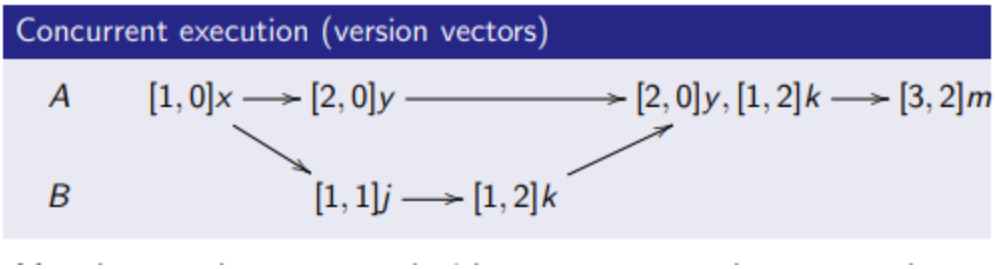
We can have concurrent actions, that's why we have the partial order. The question is: how can we merge multi-value registers?

Implementing

The concurrency is tracked with version vectors.



By having a concurrent execution, we need to merge.



This merge will produce a posterior version.