Consistency

Paulo Sérgio Almeida

February 2023

Universidade do Minho

Abstractions

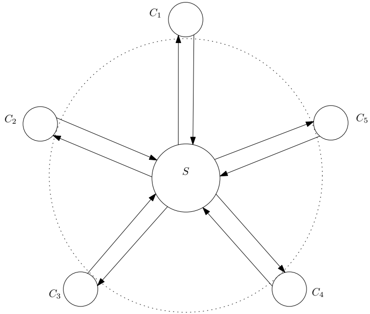
- In a process we can have abstract sequential datatypes
- In a distributed system we can also have abstract datatypes
  - registers (read, write)
  - counters (inc)
  - sets (add, remove)
- But what can we count on using such abstractions?
- Many possible design choices and tradeoffs
- Motivated by distribution, architecture, goals
- Looking at some architectures helps understanding the issue

Architectures

What is a consistency model

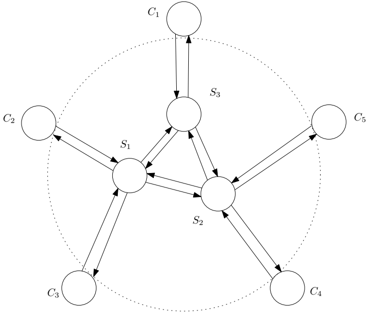
Some consistency models

Single server



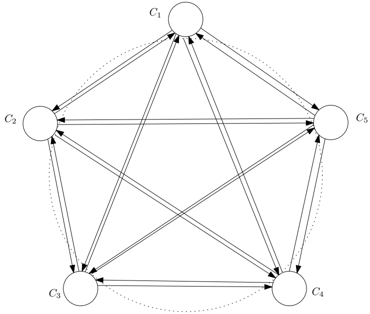
- A single server, used by all clients
- Easy to emulate sequential datatype: server serves in sequence
- Single server a single point of failure (SPOF)
- Does not scale to many clients

Multiple replicated servers



- Multiple servers run symmetrical distributed algorithm
- Each replicates part or the whole data
- Each client picks one server, and possibly change server
- Can possibly scale to many clients and avoid a SPOF
- Costly to emulate a sequential datatype

Pure peer-to-peer system



- There are no “servers”
- Each “client” node maintains full replica of state
- Each node talks directly to (possibly a subset of) others
- Can possibly scale to many clients and avoid a SPOF
- Costly to emulate a sequential datatype

Architectures

What is a consistency model

Some consistency models

An ideal world vs actual possible outcomes

- Everything would be easy if operations
  - executed instantly
  - had effects propagated globally, instantly
- The above is physically impossible
- Not relevant only for distributed systems
- Example: in a Java program, two threads reading and writing global variables, initially $a = b = 0$, without synchronization

```
(thread 1)      (thread 2)
  a = 1;          b = 1;
  x = b;          y = a;
```

- Is outcome $x = 0$ and $y = 0$ possible?
  - cannot be explained by "things happening in some order"
  - we would prefer that to be impossible – for reasoning
  - why would JVM designers make it possible?
- A consistency model describes possible outcomes

Why relax consistency in distributed systems

- In a distributed system (remember example architectures)
  - messages take time to propagate
  - data may be replicated in several nodes
  - replicas receive updates concurrently
  - there can be network partitions
- If we try to emulate sequential datatype semantics
  - must slow everything down, to avoid replicas diverging
  - network partitions may cause unavailability (no response)
- Practically, we must make some tradeoffs, relaxing consistency
  - must allow more possible, strange, outcomes
  - to allow better performance and availability

Processes, operations, local order

- Each process issues operations
  - operations of each process are totally ordered
  - process is a logical entity; can be thread, OS process, node
  - we assume, reasonably, a finite number of processes
- Session order (so):
  - session: operations from the same process
  - $a \xrightarrow{\text{so}} b$ if a is issued before b, by the same process
  - session order is itself a partial order
- Operations are broadly divided into
  - writes: update operations, that change state, may also read it
  - reads: query operations, only read

Global Orders

- Visibility (vis)
  - $a \xrightarrow{\text{vis}} b$ if effect of a visible to process performing b
  - example: read b seeing value written by write a
  - characterizes how updates are propagated and seen
- Happens-before (hb)
  - $a \xrightarrow{\text{hb}} b$ if a is in the potential causal past of b
  - analogous to classic one, for observed behavior; "semantic hb"
  - combines session order and visibility

$$\mathsf{hb} \doteq (\mathsf{so} \cup \mathsf{vis})^{+}$$

- Arbitration (ar)
  - total order on all operations
  - explains how conflicts are resolved; better if not needed
  - should at least be compatible with visibility

$$\mathsf{vis} \subseteq \mathsf{ar}$$

  - preferably, we want *causal arbitration*, compatible with hb

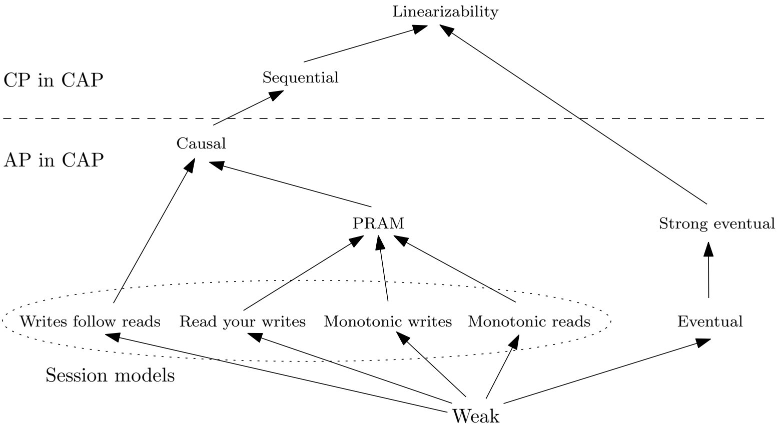
$$\mathsf{hb} \subseteq \mathsf{ar}$$

Architectures

What is a consistency model

Some consistency models

A taxonomy of some consistency models



Session models

- Basic desirable guarantees for weakly replicated data
- Each process, individually, should "see and propagate progress"
- Otherwise, very weird things will happen
- The 4 sessions guarantees are

  RYW - read your writes
  MR - monotonic reads
  MW - monotonic writes
  WFR - writes follow reads

Session models – read your writes

- RYW: reads reflect previous writes from the same process

$$\mathsf{RYW} \doteq \forall o,\ \mathsf{write}\, w \cdot w \xrightarrow{\mathsf{so}} o \Rightarrow w \xrightarrow{\mathsf{vis}} o$$

- If it doesn't hold, it confuses clients/applications
  - a process updates a database
  - it then reads from database
  - the update is missing
- Reads restricted to replicas that include previous own writes
- Easy to hold with *server affinity*
  - if client process can stick to same server

Session models – monotonic reads

- MR: successive reads see at least the same writes

$$\mathsf{MR} \doteq \forall o, o', \mathsf{write}\, w \cdot w \xrightarrow{\mathsf{vis}} o \wedge o \xrightarrow{\mathsf{so}} o' \Rightarrow w \xrightarrow{\mathsf{vis}} o'$$

- Example, with MR
  - a distributed database maps keys to records
  - records are inserted or updated by writes, never removed
  - a read returns a record for a given key
  - a subsequent read always finds that key present
  - it does not matter who is doing writes

Session models – monotonic writes

- MW: writes by same process are propagated in session order

$$
\begin{aligned}
\mathsf{MW} \doteq \forall o, \text{writes}\, w, w' \cdot \\
w \xrightarrow{\mathsf{so}} w' \wedge w' \xrightarrow{\mathsf{vis}} o \Rightarrow \\
w \xrightarrow{\mathsf{vis}} o \wedge w \xrightarrow{\mathsf{ar}} w'
\end{aligned}
$$

  where $w \xrightarrow{\mathsf{ar}} w'$ is redundant in the normal case when $\mathsf{so} \subseteq \mathsf{ar}$
- Guarantee useful also to other processes
- Example, with MW
  - consider a distributed file system
  - a process writes a new version of a library file
  - new version upward compatible and extends features
  - the process updates executable using new library features
  - if a process reads updated executable, it reads updated library

Session models – writes follow reads

- WFR: writes after a read are propagated after observed writes

$$
\begin{aligned}
\mathrm{WFR} \doteq \forall o, \text{writes}\, w, w', \text{read}\, r \cdot \\
w \xrightarrow{\text{vis}} r \wedge r \xrightarrow{\text{so}} w' \wedge w' \xrightarrow{\text{vis}} o \Rightarrow \\
w \xrightarrow{\text{vis}} o \wedge w \xrightarrow{\text{ar}} w'
\end{aligned}
$$

- Guarantee useful also to other processes
- A form of causality propagation
- Example, with WFR
  - a process reads a database record
  - notices that a field has a wrong value
  - writes the updated record
  - the updated version is propagated after original
  - replaces the previous version at all replicas

PRAM (or FIFO) - Pipelined Random Access Memory

- PRAM: writes seen in session order, own writes immediately

$$
\begin{aligned}
\mathsf{PRAM} \doteq \forall o, o', \text{writes } w, w' \cdot \\
& (w \xrightarrow{\mathsf{so}} o \Rightarrow w \xrightarrow{\mathsf{vis}} o) \wedge \\
& (w \xrightarrow{\mathsf{vis}} o \wedge o \xrightarrow{\mathsf{so}} o' \Rightarrow w \xrightarrow{\mathsf{vis}} o') \wedge \\
& (w \xrightarrow{\mathsf{so}} w' \wedge w' \xrightarrow{\mathsf{vis}} o \Rightarrow w \xrightarrow{\mathsf{vis}} o)
\end{aligned}
$$

- Equivalent to combination of RYW, MR, MW

$$\mathsf{PRAM} \equiv \mathsf{RYW} \wedge \mathsf{MR} \wedge \mathsf{MW}$$

- As if writes from each process propagate in FIFO order
- Writes from different processess may be seen in different orders
- No global total order exists

Causal consistency

- One of the most important consistency models
- Broadly speaking
  - all potential semantic causal past is available to an operation
  - it is the strongest model that remains available under partitions
- The essential aspect

$$\mathsf{vis} = \mathsf{hb}$$

  - write w that potentially influences w' becomes visible first
  - a read operation sees all writes that potentially influence it

Causal consistency – arbitration

- For some datatypes no order is needed at all
  - aggregation-like semantics, commutative and associative
  - just need to have the set of visible operations
  - example: counters, with an "increment" operation
- For most datatypes we need order
  - but not necessarily arbitration (a total order)
  - may be enough the set of maximal operations under hb
    - set of more recent concurrent operations
  - many conflict-free replicated datatypes
- In some cases we can resort to a causal arbitration ($\mathsf{hb} \subseteq \mathsf{ar}$)
  - example: classic registers (memory) with read/write
  - use hb-compatible "last-writer-wins" (not physical time)

Causality does not imply convergence

- All previous consistency models allow replicas diverging
- And even never converging again
- Even with causal consistency, replicas may diverge forever
- Example: causal memory
  - two processes concurrently write `x = 1` and `x = 2`
  - one process may read forever `x` as 1 and another `x` as 2
- Something more is desirable using a weak consistency model
  - convergence

Eventual visibility

- First ingredient for convergence: eventual visibility
  - each write becomes eventually visible, i.e.,
  - a write can take some time to become visible ...

    ... but sooner or later it will become visible everywhere

Eventual visibility

For any write w in an infinite history

$$\left|\{o \mid w \stackrel{\text{vis}}{\not\longrightarrow} o\}\right| < \infty$$

Convergence

- Strong Convergence
  - two reads that see same write operations return same value

$$\text{StrongConvergence} \doteq \forall \text{ reads } r, r' \cdot \\ \{\text{write } w \mid w \xrightarrow{\text{vis}} r\} = \{\text{write } w \mid w \xrightarrow{\text{vis}} r'\} \Rightarrow \\ \text{value}(r) = \text{value}(r')$$

- Convergence
  - if we keep issuing reads that see same write operations ...
    ... the reads will eventually return same value
- The best, and what normally happens is strong convergence
  - with automatic conflict resolution, as writes become visible
  - why would we want otherwise?
  - only if conflict resolution was manual or postponed

Eventual consistency and strong eventual consistency

- Strong eventual consistency (SEC) means having both
  - eventual visibility
  - strong convergence
- Eventual consistency (EC) means having both
  - eventual visibility
  - convergence
- Remarks
  - EC should have been defined as the SEC variant
  - using "strong" in the name is a source of confusion with strong consistency models

Combining order and convergence

- Order and convergence guarantees can be combined
- Causal+ consistency is the combination of
  - causal consistency
  - strong convergence

> Ensuring the combination of causal consistency and strong convergence is the standard design goal for scalable highly available replicated datatypes

Strong consistency models

- Sequential consistency and Linearizability
- Same semantics as if all operations occurred in a sequence
- As if there is some arbitration such that
  - arbitration is compatible with session order
  - semantics as if all operations were issued by a single process

$$\mathsf{so} \subseteq \mathsf{vis} = \mathsf{ar}$$

- For sequential consistency "real time" does not matter
- Linearizability ensures a stronger guarantee
  - as if each operation occurs instantly
  - in a point in time between invocation and return time

The CAP theorem

The CAP theorem

From strong Consistency, Availability, and Partition tolerance, we can have at most two

- As partitions happen, and cannot be avoided, we can either design AP systems or CP systems
- CP systems
  - aim typically for linearizability
  - may become unavailable (stall) under network partitions
- AP systems
  - operations remain available, even when there are partitions
  - at most we aim for causal consistency and strong convergence
  - AP more suitable for truly global large scale systems