Distributed Computing

A-02. Transactions, ACID and CAP

Transactions

- A transaction for us is an independent modification in a system that stores data
  - Database, file system, …
- While they may change **several** parts of the system at once, we think about them as a single modification
- When money is transferred, it is “simultaneously” removed from one account and put in another one
- A directory is removed
- A new version of a file is saved

ACID Properties (1)

- **Atomicity**, **Consistency**, **Isolation**, **Durability**
- A classic set of properties to implement transactions
- Makes it **easier** to think about how the system behaves
- Implemented in 1973 (Grey and Reuter 1993, page 42), even though the acronym was coined 10 years later (Härder and Reuter 1983)

ACID properties (2)

- **Atomicity**: each transaction is treated as a single unit, that is it either succeeds or fails completely
  - E.g., If money is taken from my account, it gets to the destination
  - E.g., If I save a new version of a file, nobody will see a “half-written” version of it
- **Consistency** (Correctness): the system remains in a valid state
  - E.g., All accounts have non-negative balance
  - E.g., A non-deleted directory is reachable from the root

ACID Properties (3)

- **Isolation**: even if transactions may be run concurrently, the system behaves as if they’ve been running sequentially
  - Transactions are seen as “ordered”
- **Durability**: Even in case of a system failure, the result of the transaction is not lost

The CAP Theorem

- Proposed as a conjecture by Fox and Brewer in 1999
- Proven as a theorem by Gilbert and Lynch in 2002
- In a system (that allows transactions), you cannot have all of **consistency**, **availability** and **partition tolerance**

C, A and P

- **Consistency**: every read receives the most recent write or an error
- **Availability**: every request receives a non-error response
- **Partition Tolerance**: the system keeps working even if an arbitrary number of messages between the nodes of our distributed system is dropped

The Easy Proof

- Suppose the system is partitioned in two parts, G_1 and G_2: no communication happens between them
- A write happens in G_1
- A read happens in G_2
- The result of the write is not accessible from G_2, so one of these happens:
  - The system returns an error (we lose **availability**)
  - The system returns old data (we lose **consistency**)
  - The system doesn’t reply (we lose **partition tolerance**)

Questions!

- app.wooclap.com/DC24UNIGE

The Not-So-Obvious Consequences

- In any distributed system, you have a trade-off:
  - Either (part of) your system will be **offline** until the network partition is resolved
  - Or you will have to live with inconsistent and stale data
- Later in the course, we’ll dive in work that explores this tradeoff. Distributed systems are very often about tradeoffs!
  - A piece about how this impacted system design by Brewer in 2012

Examples of Non-ACID Systems

- Can you think of systems that can work with inconsistent functionality?
  - GIT (conflicts)
  - DNS
  - Social networks
  - NoSQL databases