Advanced Enterprise Computing - Lecturenotes SoSe2016

Julius Hülsmann

20. Juli 2016

Inhaltsverzeichnis

1 Repetition

1.1 ACID

Atomiticity: Either the entire Transaction is executed or it is completely aborted.
Consistency: does not mean Data-Consistency but that the transaction produces consistent changes.
Isolation: Transactions are isolated from one another
Durability: Once the transaction is ready (commits) it remains.

Both the Atomitcity and the Isolation are managed by the **Transaction Manager**

- Aquires locks on behalf of the transactions

- guarantees serializable execution. Therefore, the scheduler only needs to enforce 2PL behavior because serializability is automatically guaranteed in case the transaction follow 2PL

name	protocol's name	examples of implementations
Atomacity	atomic committement protocol	2PC
Isolation	concurrency control protocol	2PL, Snapshot isolation

1.2 CAP

CAP = Consistency vs. Avaiability in case of Partitioning (Replication) Either one has to choose consistency or availability.
Gives information on the system's behavior in case of a system error.

1.3 PACELC

PACELC: partitioning: Avaibaility/Consistency else Latency/Consistency
In case an update request arrives and the data is replicated and the system is working properly, there is a time difference between the moment the first replica receives the update and the other replicas are informed. Now it's the question whether to

1. immeadiately commit the new data *(chose the least latency)* or

2. whether not to respond until the 2nd replica respons *(chose consistency)*

1.4   Consistency

2 Dimensions:

- **Staleness** (how much is a given replica lagging behind) vs. **Ordering** (how much does the operation serializable order deviate among replicas)

- **Data-centric** and **Client-centric** consistency

1.4.1   Ordering and Data-centric consistency

- Sequential Consistency

- Causal consistency

- Eventual Consistency

1.4.2   Ordering and Client-centric consistency

- Monotonic Reads (never receive older values than previously read)

- Read-my-writes (not older than previously written)

- Write follows reads (only update Replicas that have at least got read entity)

- Monotonic Writes (update of one client: always executed sorted by time)

2   Replication and State Management (25.04. - 09.05.)

2.1   Motivation and Background

2.1.1   Replication

Definition - Replication   Process of maintaining multiple Copies of an Entity (Data / Process / File ...)

Advantages of Replication in General

- *System Availability / Fault tolerance* in case
  A   Server fails
  B   Data is corrputed.

- *Performance / Scalability*
  A   Workloads are spread across distributed Replicas
  B   Geodistribution for processing demands in client's proximity

Disadvantages of Replication in General

- Consistency vs. Performance

Kinds of Replication   There are the following three different decisions one needs to take for developing a suitable Replication design:

1. „Phyiscal Replication"(A), (B) or (C)

2. Replication Strategies (Synchronous/Asynchronous PrimaryCopy/Update Everywhere)

3. Where to put the Replicas (Geo-distributed?)

decision 1) Replication   In general there are the following kinds of „physical" Replication. We do only consider (B).
!!!!!!!!!!!!!!!!ueberpruefen am Ende!!!!!!!!!!!!!!!!

decision 2) Replication Strategies
Synchronous / eager vs. Asynchronous / lazy
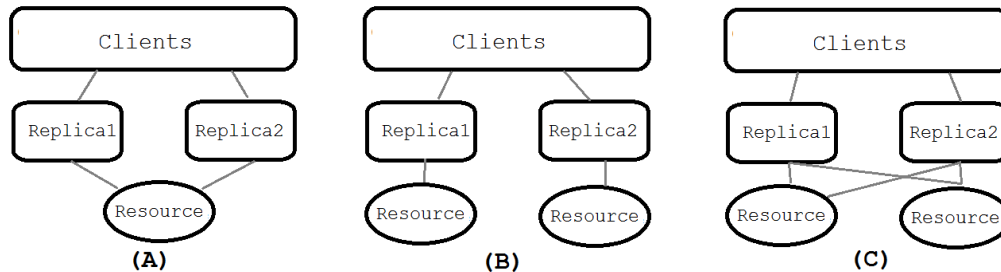Primary Copy / master vs. Update Everywhere / group

Abbildung 1: (A) Improves the availability only in case the Server Replicas use a replica cache coherence mechanism (B) Improves the availability. Usually ment by the term „Replication".

decision 3) Replicas' location There are permanent Replicas, they can be geo-distributed or within the LAN
Server-initiated (e.g. Push-Cache) or Client-initiated (e.g. Cache) temporary Replicas exist, too.
Push-Cache: Server record request-history

What happens to ACID in case of Replication? Atomicity can be guaranteed using 2PC (but expensive) Problem: Serialization order must be the same at all replicas.
Synchronous

2.2 managing Replication

Synchronous Update Everywhere Protocol Assumption: All sites (Replicas) Contain the same data. Behavior if Transaction is to be executed

- Local use of 2PL for the following steps:

- READ only one site, in case the reading fails (timeout), read another copy

- WRITE at all sites (distributed locking protocol). This means that all copies of the data item need to lock the item (REQUEST, OBTAIN LOCK, ACK)
.
IF one site rejects, ABORT.
ADD All site not responding to a list of *missing writes*.

- VALIDATE (=commit) the transaction at the end, this means

IF NOT ALL the servers *missing writes* are down: ABORT
IF NOT ALL the servers that *accepted* are still available: ABORT
OTHERWISE Commits

$\Rightarrow$ Guarantees behaviour like if the sites were not replicated.

$\Rightarrow$ Execution is serializable

$\Rightarrow$ all Reads access the latest version

Extensions for coping with failures
1. Site failure (reduces the avalability)
2. Behavior after the Site that failed is online again (outdated data available)
Optimization: Most ideas based on Quorums

Asynchronous Primary Copy

- BOT: Primary Copy receives transaction

- EOT: Primary Copy applies transaction (2PL locally) and COMMITS

- The changes are propagated to all Replicas

Asynchronous Update Everywhere

- BOT, W, R, ..., EOT executed locally (2PL locally) (locally committed)

- Changes are propagated to all other replicas

- Reconciliation (Last update, site priority, value priority, try to merge, rm not important one, priority shemes

Quorums   kind of a middleground between syncrhonous and Asynchronous updates.
Reads contact more than one Replica
Write contacts a quorum of Replicas.
Rules:
Read at least 1 Replica that has received the latest update: $R + W > N$,
The minimum amount of writes must be greater than half of the amount of replicas $\frac{W}{2} > N$
Quorums that don't follow these rules are called **sloppy Quroums**. (Dynamo + Cassandra)
Used to trade off (+) read and (-) write latency

Different views on the subject   The solution to Replication strategies in *Database-POV* and *Distributed Systems-POV* differ, but have converged. Distributed System

- Set of Services implemented by Server Processes, invoked by client processes

- Each server has got a local state

- Group of servers / group communication – helps to reduce complexity

- Group communciation primitives: provide 1toMany communication. Example: Atomic Broadcast (ACAST), View-Syncrhonous Broadcast (VSCAST)

Active Replication   ABCAST guarantees atomacity, sent to everybody All Servers execute the transaction in the order they arrive Everybody informs the client (which typically takes the first answer).
+ Simple + failure-transparency; Failures completely hidden from client - determinism constraint (no multicore servers) c - complexity shifted to middleware layer (communication)

Passive Replication   Client -¿ Primary which executes the Transaction and sents the result to the backups
Connunication has to guarantee right order (VSCAST)
By doing so, no determinism constraint is necessary on the execution of invocations.
1 Client sents request 2 Primary applies directly 3 primary informs other servers (VSCAST) 4 Primary sents answer to the client
+ tolerates non-deterministic servers (Threaded) + Little processing power used - High reconfiguration cost if primary fails

Semi-Active Replication   Combination of both
Each time Replicas have to do a non-deterministic decision, one replica (called Leader) broadcasts the result to the other replicas(followers).

2.3   Implications of Replication

7 Lecturenotes

Lecture 05? start @ 81 Für Donnerstag paper mitbringen und Paxos anschauen.
2016-05-09

Paxos (Represent as State-machine) - P. 77
Proposer
Phase 1 - Proposer choses Number largr than any value chosen before by
Propposer. - Broadcast the integer *prepare(n)*, e.g. prepare(50)
Acceptors a) Not respond at all b) *recject* Reject, in case a higher value has
been accepted. 50 ¡ something b) *prommise(n)* in case 50 ¿ everything. Also
Send everything that has already been accepted.
If prposer receives majority of prommise respons, -¿ proceed to Phase 2 ELSE
-¿ Phase 1
Phase 2 - Check whether any ¡n, value¿ have been returned. - YES: take max
n's value - accept (n, value)

Xtensions Paxos   **Multi-paxos** Determine Leader once Stay in phase 2,
attatch the leader identifier Leader is the one to accept values
Purpose: Optimize Speed (get rid of the first phase, Master-Slave setup)
Fast Paxos
Generalized Paxos - Assumption: The execution order does not matter.

CRDT Conflict free / Communitive replicated Datatypes
Some operations are commutative, others not.
State- Based vs. Operation based.
theoretically it is possible to converge them but ... practice
IDEA INTEGER - example: e.g. not store int values but operations (increment / decrement))
SET - example
State - based Set

8 Begriffe und Abkürzungen

Replication Strategy to maintain mutiple copies of an entity on multiple
Servers.

Replica

CRDT *conflict-free replicated data*

Paxos

Commit In case a Transaction commits, it is ready.

Concurrency control protocol guarantees isolation of Transactions

2PL Two phase locking (one concurrency control protocol)

Snapshot Isolation other cuncurrency control protocol implementation

atomic commitement protocol guarantees atomaticity

2PC Two phase Commit

Transaction Manager Middleware Component; Manages Atomacity and Isolation of Transactions

ACID Atomacity + Consistency + Isolation + Durability

serializability a plan of executing multiple transactions in pseudo- parallel is called serializable in case the parallel execution comes to the same result as executing the transactions one after the other

distributed locking protocol 2PL z.b.? ??? Paxos??ß