Real-world consistency explained
or the challenges of modern persistence

Uwe Friedrichsen (codecentric AG) – Velocity – London, 19. October 2017
Some kudos first ...
A lot of this talk was inspired by the great posts of Adrian Colyer especially by his blog series "Out of the fire swamp"

see [Col], [Col2015a-c]
RDBMS  ACID
RDBMS

- “One database to rule them all”
- Good all-rounder
  - Rich schema
  - Rich access patterns
- Designed for scarce resources
  - Storage, CPU, Backup are expensive
  - Network is slow
- Shared database
  - Replication was expensive
  - Licenses were expensive
  - Operations were expensive
  - Easy integration model
  - “Strange attractor”
  - Accidental central integration hub
  - Data spaghetti
• Atomicity
• Consistency
• Isolation
• Durability

• Great programming model
  • No temporal inconsistencies
  • No anomalies
  • Easy to reason about

• But reality often is different!
  • ACID does not necessarily mean “serializability”
  • Databases often run at lower consistency levels
  • Anomalies happen
  • Most developers are not aware of it
ANSI SQL

Anomalies

• Dirty write (P0): \( w_1[x]...w_2[x]...(c_1 \text{ or } a_1) \)
• Dirty read (P1): \( w_1[x]...r_2[x]...(c_1 \text{ or } a_1) \)
• Fuzzy read (P2): \( r_1[x]...w_2[x]...(c_1 \text{ or } a_1) \)
• Phantom read (P3): \( r_1[P]...w_2[y \text{ in } P]...(c_1 \text{ or } a_1) \)

Isolation levels

<table>
<thead>
<tr>
<th>Isolation level</th>
<th>Dirty write</th>
<th>Dirty read</th>
<th>Fuzzy read</th>
<th>Phantom read</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read uncommitted</td>
<td>Not possible</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Read committed</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Repeatable read</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Serializable</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
</tr>
</tbody>
</table>

See [Ber+1995]
Extended anomaly model

- Dirty write (P0): \( w_1[x] \ldots w_2[x] \ldots (c_1 \text{ or } a_1) \)
- Dirty read (P1): \( w_1[x] \ldots r_2[x] \ldots (c_1 \text{ or } a_1) \)
- Lost update (P4): \( r_1[x] \ldots w_2[x] \ldots w_1[x] \ldots c_1 \)
- Lost cursor u. (P4C): \( r_c_1[x] \ldots w_2[x] \ldots w_c_1[x] \ldots c_1. \)
- Fuzzy read (P2): \( r_1[x] \ldots w_2[x] \ldots (c_1 \text{ or } a_1) \)
- Phantom read (P3): \( r_1[P] \ldots w_2[y \text{ in } P] \ldots (c_1 \text{ or } a_1) \)
- Read skew (A5A): \( r_1[x] \ldots w_2[x] \ldots w_2[y] \ldots c_2 \ldots r_1[y] \ldots (c_1 \text{ or } a_1) \)
- Write skew (A5B): \( r_1[x] \ldots r_2[y] \ldots w_1[y] \ldots w_2[x] \ldots (c_1 \text{ and } c_2 \text{ occur}) \)

see [Ber+1995]
## Extended isolation level model

<table>
<thead>
<tr>
<th>Isolation level</th>
<th>Dirty write</th>
<th>Dirty read</th>
<th>Cursor lost update</th>
<th>Lost update</th>
<th>Fuzzy read</th>
<th>Phantom read</th>
<th>Read skew</th>
<th>Write skew</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read uncommitted</td>
<td>Not possible</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Read committed</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Cursor stability</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Sometimes possible</td>
<td>Sometimes possible</td>
<td>Possible</td>
<td>Possible</td>
<td>Sometimes possible</td>
</tr>
<tr>
<td>Repeatable read</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Possible</td>
<td>Not possible</td>
<td>Not possible</td>
</tr>
<tr>
<td>Snapshot</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Sometimes possible</td>
<td>Not possible</td>
<td>Possible</td>
</tr>
<tr>
<td>Serializable</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
<td>Not possible</td>
</tr>
</tbody>
</table>

See [Ber+1995]
## Default & maximum isolation levels

<table>
<thead>
<tr>
<th>Database</th>
<th>Default</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actian Ingres 10.0/10S [1]</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Aerospike [2]</td>
<td>RC</td>
<td>RC</td>
</tr>
<tr>
<td>Clustrix CLX 4100 [4]</td>
<td>RR</td>
<td>RR</td>
</tr>
<tr>
<td>Greenplum 4.1 [8]</td>
<td>RC</td>
<td>S</td>
</tr>
<tr>
<td>IBM DB2 10 for z/OS [5]</td>
<td>CS</td>
<td>S</td>
</tr>
<tr>
<td>IBM Informix 11.50 [9]</td>
<td>Depends</td>
<td>S</td>
</tr>
<tr>
<td>MySQL 5.6 [12]</td>
<td>RR</td>
<td>S</td>
</tr>
<tr>
<td>MemSQL 1b [10]</td>
<td>RC</td>
<td>RC</td>
</tr>
<tr>
<td>Oracle 11g [14]</td>
<td>RC</td>
<td>SI</td>
</tr>
<tr>
<td>Oracle Berkeley DB [7]</td>
<td>S</td>
<td>S</td>
</tr>
<tr>
<td>Postgres 9.2.2 [15]</td>
<td>RC</td>
<td>S</td>
</tr>
<tr>
<td>SAP HANA [16]</td>
<td>RC</td>
<td>SI</td>
</tr>
<tr>
<td>ScaleDB 1.02 [17]</td>
<td>RC</td>
<td>RC</td>
</tr>
<tr>
<td>VoltDB [18]</td>
<td>S</td>
<td>S</td>
</tr>
</tbody>
</table>


Table 1: Default and maximum isolation levels for ACID and NewSQL databases as of January 2013.
Wrap-up – Past

• The relational model is a good tradeoff
• ACID makes a developer's life easy
• Yet, we often live (unknowingly) with less than serializability
### Cloud

- Self Service
- Elasticity
- Pay per use
- Great resource provisioning model
- Improves
  - Autonomy
  - Response time (lead time)
  - Elasticity
  - Cost efficiency (if done right)
- Trade-offs
  - Scale out ("distributed hell")
  - Reduced availability of individual resources
“Microservices are the mapping of organizational autonomy to software architecture”

- Limited in scope
- Self-dependent
- Loosely coupled

Improves
- Autonomy
- Response time (if done right)
- Elasticity

Trade-offs
- Higher design effort
- Harder to operate
- Distributed by default

Shared nothing
- No shared data
- No cross-service coordination
NoSQL

- Extension of the storage solution space
  - Before NoSQL RDBMS and file system were predominant solutions
  - NoSQL tries to fill the gaps

- New options
  - Scalability (Volume & Velocity)
  - Relaxed schema
  - Availability in cloud environments

- Trade-offs
  - CAP Theorem
  - Capabilities and limitations often poorly understood
• Response to CAP theorem
  • “Relax temporal constraints in exchange for better availability”

• Improves
  • Scalability
  • Availability

• Trade-offs
  • Temporal inconsistencies and all kinds of anomalies become visible
  • Very hard programming model

• Key readings
  • [Bre2000] introduced CAP and BASE
  • [Hel2007] “defined” boundaries of eventual consistency for years
  • [Sha+2011] introduced CRDTs, improving convergence guarantees in eventually consistent systems
About polyglot persistence and choosing the right database
The 8 dimensions of storage

- Data Scalability (amount of data)
- Transaction Scalability (access rate)
- Latency (response time considering scalability)
- Read/Write Ratio (variability of r/w mix considering scalability)
- Schema Richness (variability of data model)
- Access Richness (variability of access patterns)
- Consistency (data consistency guarantees)
- Fault Tolerance (ability to handle failures gracefully)
File system

Data scalability
Transaction scalability
Fault tolerance
Consistency
Access richness
R/W ratio
Schema richness
Latency
Data scalability
Transaction scalability
Schema richness
Access richness
R/W ratio
Consistency
Fault tolerance
Latency

NoSQL
Filling the blind spots of RDBMS and file systems

RDBMS & file system
Data scalability
Transaction scalability
Schema richness
Access richness
Consistency
Fault tolerance
Latency
R/W ratio

RDBMS-FS-Cassandra
There is no “one size fits all” and no drop-in replacement for your GORDB *

* good ol’ relational database
About eventual consistency
NoSQL databases ...

- ... often do not guarantee any consistency out of the box
- ... need to be configured properly for eventual consistency
- ... sometimes can provide higher consistency guarantees
- ... sometimes even ditch eventual consistency for better availability
  - Minimum default consistency level to expect: single write/single entity/single node
You will experience temporary inconsistencies and anomalies at the application level
Inconsistencies across replica sets

Typical measures

- Read your writes
- Read from master
- Quorum based reads and writes
  - Warning: This will *not* give you strict consistency!
Siblings

Typical measures

- Return all siblings, leave decision to client
- Resolver ("Read repair")
- Conflict-free Replicated Data Types (CRDT)
  - Great stuff, but brain twister and does not work for all data types
Performance hit for random access

Usually no reliable and efficient secondary indices possible
  • Efficient and reliable access only via primary key access

Typical measures
  • Coarse-grained entities plus denormalized associations
  • Multiple entity representations (one per access path)
  • Results in new type of inconsistencies (across entities)
  • Requires additional type of resolver (cross-entity resolver)
Thus, *really* understand your consistency options ...
... and be wary of vendor promises
Wrap-up – Present

- IT goes distributed ("scale out")
- NoSQL fills the empty spots in the storage solution space
- BASE transactions imply a very hard programming model
So, can I only choose between “ACID” and “BASE”?
Remember, that “ACID” does not necessarily mean serializability?
Repeatable read
One-copy serializability
Strong one-copy serializability

Read committed
Read uncommitted
Repeatable read

Cursor stability
Monotonic atomic view

Snapshot isolation
Predicate cut isolation
Item cut isolation
Writes follow reads
Monotonic reads
Monotonic writes
Read your writes

Monotonic reads
Monotonic writes

Causal consistency
PRAM

linearizable
regular
safe
recency

Highly available transaction model
Sticky available transaction model
Non-HA transaction model

see [Bai+2014a]
There are a lot of consistency options to choose from between “ACID” and “BASE”
But ...
The old model assumed the work would be processed in exactly one order of execution. There was a default “single system of record” form of isolation provided by the classic database system running at the primary. This single history allows for a low-level READ and WRITE semantic that depends on “replaying history”.

In this new [distributed] world, history cannot be exactly replayed and we must count on the ability to reorder the work. This means that we cannot completely know the accurate state of the system. It also means we must move the correctness and reordering semantics up from being based on system properties (i.e. READ and WRITE) to application based business operations.

see [Hel+2009]
It is no longer sufficient to understand and reason about distributed application consistency at the level of data store access. Instead you need to understand and reason about distributed application consistency at the level of application operations.
1. You need to analyze the consistency requirements of the application carefully based on the business requirements.

2. The application (usually) needs to contribute explicitly to the implementation of the required consistency model.
Current research explores the “frontiers”

- "HAT, not CAP: Towards Highly Available Transactions“ [Bai+2013b]
  HA causal consistency in distributed systems
- "Scalable Atomic Visibility with RAMP Transactions" [Bai+2014b]
  New isolation level “atomic read” (stronger than MAV), implemented in HA fashion
- "Building Consistent Transactions with Inconsistent Replication" [Zha+2015]
  Low-latency distributed transactions by building a transactional application protocol on top of inconsistent replication
- "Putting Consistency Back into Eventual Consistency" [Bal+2015]
  Explicit application-level consistency using application-defined invariants on top of eventual consistent data stores
- "Implementing Linearizability at Large Scale and Low Latency" [Lee+2015]
  Implementing fast and scalable exactly-once semantics, enabling linearizable operations on top of it
- "Spanner: Google’s Globally-Distributed Database" [Cor+2012]
  “Case study” for a very different approach to scalable serializability by using hardware to solve the “time problem”
- "High-Performance ACID via Modular Concurrency Control" [Xie+2015]
  Speedup of traditional ACID transactions by grouping transactions into independent sets that can be handled concurrently
And there is more to come ...
Current memory and storage trends

- Terabyte memory computers
  Full in-memory computing of large data sets

- Storage Class Memory (SCM) / Non-Volatile RAM (NVRAM)
  Many technologies under development filling the gap between RAM and SSD

- Remote Direct Memory Access (RDMA)
  Accessing a remote machine’s RAM bypassing the CPU, allowing very low-latency remote memory access (around 2 order of magnitude faster than SSD access)

Keeping the CPU busy is no longer the core challenge

New system architectures and programming models will emerge

New consistency options not available today may also emerge

see [Coc2016], [Col2016]
Wrap-up – Future

- Lots of options to balance consistency constraints and intricacy of the programming model
- Higher consistency guarantees than “plain BASE” in distributed HA databases may become available in the near future
- Most of them will require effort on the application level
Recommendations
Across service/storage boundaries

- Don't coordinate writes across service/storage boundaries
  - Usually your design is wrong (entity-driven instead of behavior-driven)
  - Remember: DDD is about *domains*, not entities! Domains include behavior
  - The activation path of a use case should be as short as possible

- Use a relaxed temporal constraint model plus reconciliation
  - Consider applying the concepts of promise theory [Bur2005] and memory, guesses and apologies [Hel+2009]
Within service/storage boundaries

- Thoroughly analyze your storage requirements (8 dimensions)
- Thoroughly analyze your consistency requirements
- Be wary of serializability (database reality is different)
- Don't think consistency is solely a data store issue
- Don't distribute data or relax consistency without an explicit need
- Choose wisely – and have your developers in mind
Wrap-up

- Past: RDBMS and ACID
  - Great programming model
  - ACID does not necessarily mean serializability
- Present: Cloud, µServices, NoSQL & BASE
  - More options, more challenges
  - Very hard programming model
- Future: Exploring the boundaries
  - Many options between ACID and BASE
  - Often requires awareness on the application level
- Know your options!
There is no "one-size-fits-all" solution
References


References


[Col] Adrian Colyer, "the morning paper", http://blog.acolyer.org


[Col2016] Adrian Colyer, "All change please", http://blog.acolyer.org/2016/01/22/all-change-please/

References


[Hel+2009] Pat Helland, Dave Campell, "Building on Quicksand", CIDR 2009


