Real-world consistency explained

or the challenges of modern persistence

Uwe Friedrichsen (codecentric AG) – Velocity – London, 19. October 2017

@ufried

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]

Past

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

ACID

- 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):	w1[x]…w2[x]…(c1 or a1)
• Dirty read (P1):	w1[x]…r2[x]…(c1 or a1)
• Fuzzy read (P2):	r1[x]…w2[x]…(c1 or a1)
• Phantom read (P3):	r1[P]…w2[y in P]…(c1 or a1)

Isolation levels

	Dirty write	*Dirty read*	*Fuzzy read*	*Phantom read*
Read uncommitted	Not possible	Possible	Possible	Possible
Read committed	Not possible	Not possible	Possible	Possible
Repeatable read	Not possible	Not possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible	Not possible

See [Ber+1995]

Extended anomaly model

• Dirty write (P0):	w1[x]…w2[x]…(c1 or a1)
• Dirty read (P1):	w1[x]…r2[x]…(c1 or a1)
• Lost update (P4):	r1[x]…w2[x]…w1[x]…c1
• Lost cursor u. (P4C):	rc1[x]…w2[x]…wc1[x]…c1.
• Fuzzy read (P2):	r1[x]…w2[x]…(c1 or a1)
• Phantom read (P3):	r1[P]…w2[y in P]…(c1 or a1)
• Read skew (A5A):	r1[x]…w2[x]…w2[y]…c2…r1[y]…(c1 or a1)
• Write skew (A5B):	r1[x]…r2[y]…w1[y]…w2[x]…(c1 and c2 occur)

see [Ber+1995]

Extended isolation level model

Isolation level	*Dirty write*	*Dirty read*	*Cursor lost update*	*Lost update*	*Fuzzy read*	*Phantom read*	*Read skew*	*Write skew*
Read uncommitted	Not possible	Possible	Possible	Possible	Possible	Possible	Possible	Possible
Read committed	Not possible	Not possible	Possible	Possible	Possible	Possible	Possible	Possible
Cursor stability	Not possible	Not possible	Not possible	Sometimes possible	Sometimes possible	Possible	Possible	Sometimes possible
Repeatable read	Not possible	Not possible	Not possible	Not possible	Not possible	Possible	Not possible	Not possible
Snapshot	Not possible	Not possible	Not possible	Not possible	Not possible	Sometimes possible	Not possible	Possible
Serializable	Not possible	Not possible	Not possible	Not possible	Not possible	Not possible	Not possible	Not possible

See [Ber+1995]

Default & maximum isolation levels

Database	Default	Maximum
Actian Ingres 10.0/10S [1]	S	S
Aerospike [2]	RC	RC
Akiban Persistit [3]	SI	SI
Clustrix CLX 4100 [4]	RR	RR
Greenplum 4.1 [8]	RC	S
IBM DB2 10 for z/OS [5]	CS	S
IBM Informix 11.50 [9]	Depends	S
MySQL 5.6 [12]	RR	S
MemSQL 1b [10]	RC	RC
MS SQL Server 2012 [11]	RC	S
NuoDB [13]	CR	CR
Oracle 11g [14]	RC	SI
Oracle Berkeley DB [7]	S	S
Oracle Berkeley DB JE [6]	RR	S
Postgres 9.2.2 [15]	RC	S
SAP HANA [16]	RC	SI
ScaleDB 1.02 [17]	RC	RC
VoltDB [18]	S	S
RC: read committed, RR: repeatable read, SI: snapshot isolation, S: serializability, CS: cursor stability, CR: consistent read		

Table 1: Default and maximum isolation levels for ACID and NewSQL databases as of January 2013.

See [Bai+2013a]

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

Present

Cloud

µService

NoSQL

BASE

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

µService

- “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

BASE

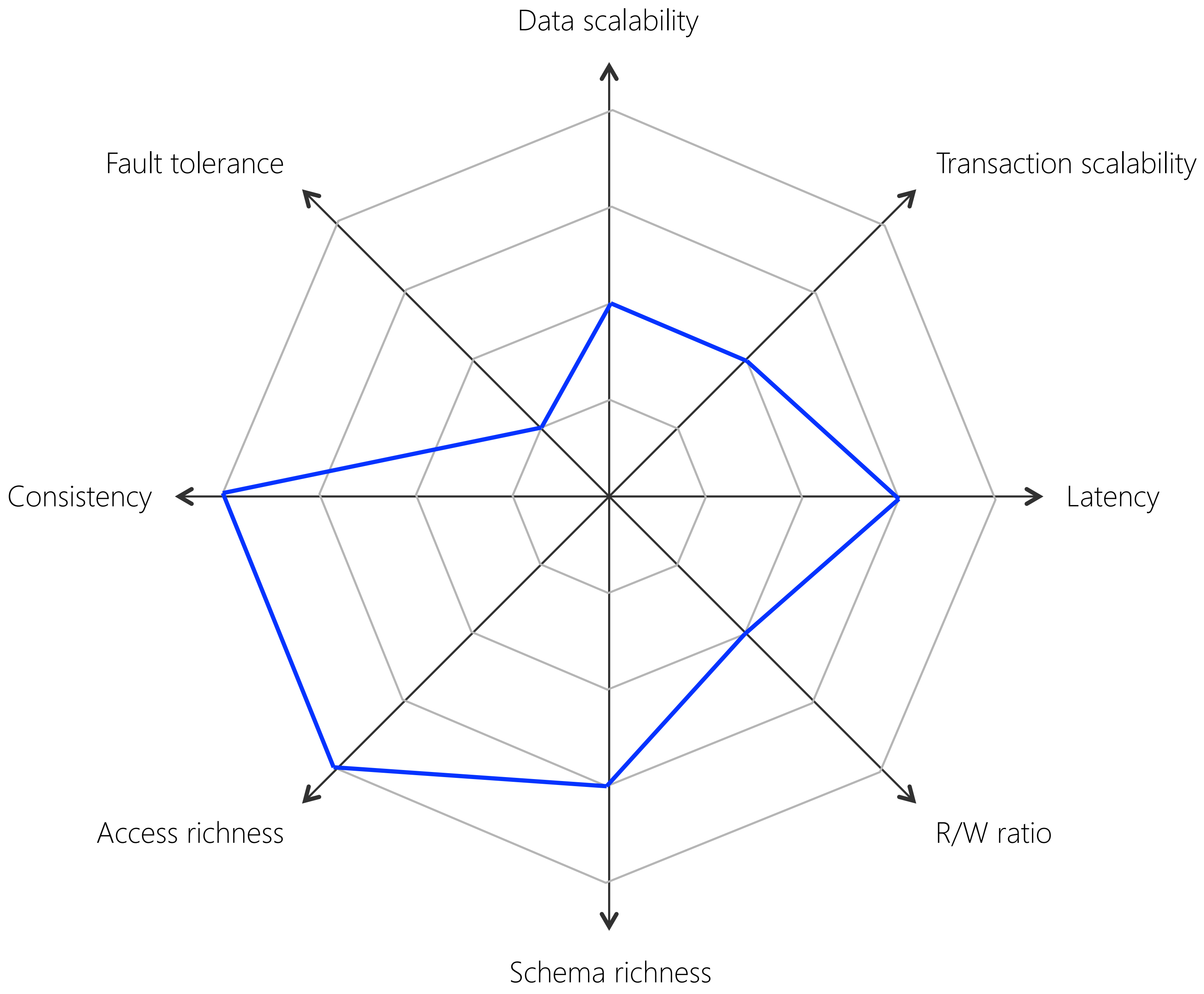
- Response to CAP theorem
  - “Relax temporal constraints in exchange for better availability”
- Improves
  - Scalability
  - Availability
- Trade-offs
  - Temporal inconsistencies and all kinds of anomalies become visible
  - Very hard programming model
- Key readings
  - [Bre2000] introduced CAP and 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

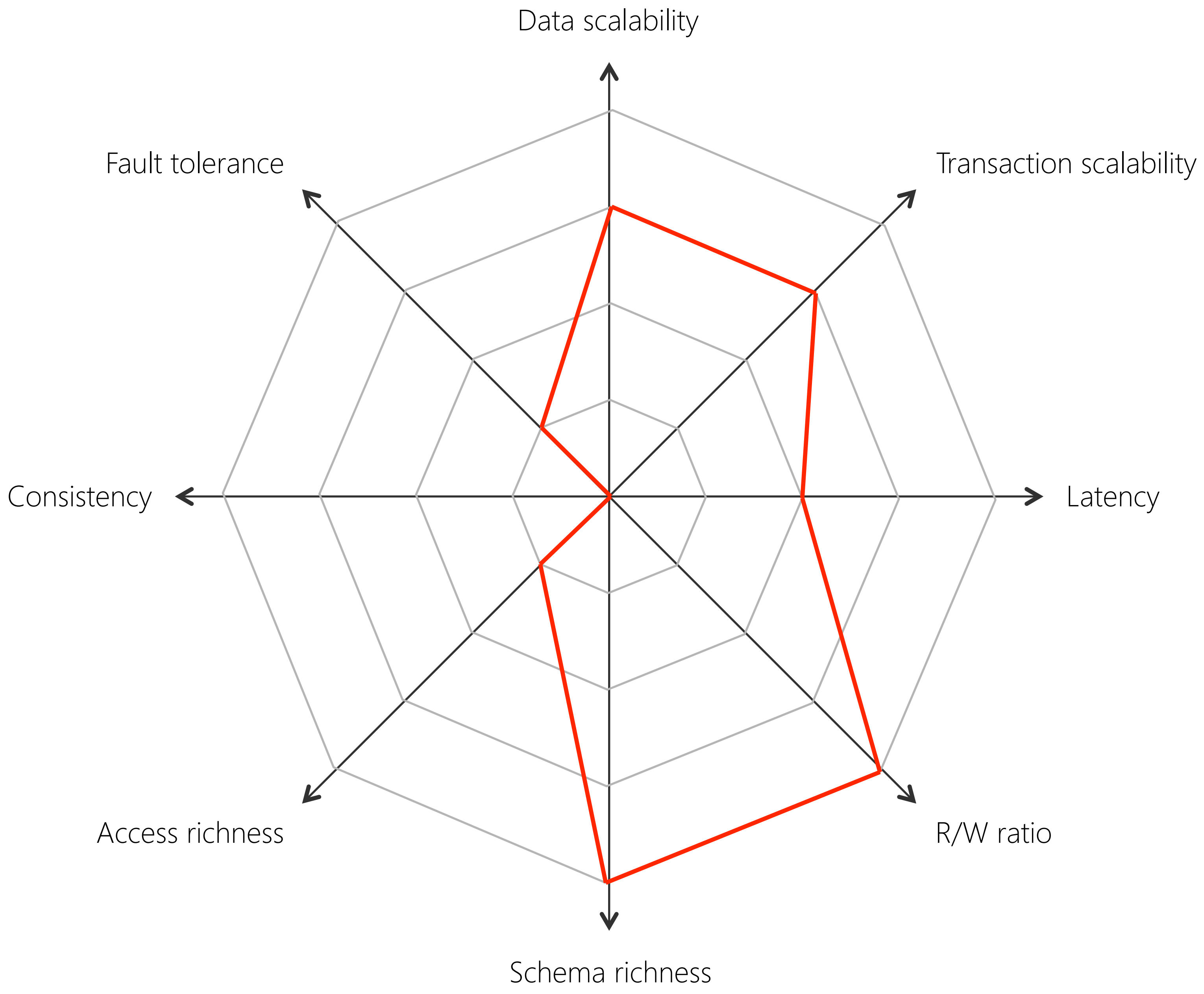
The 8 dimensions of storage

- 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)

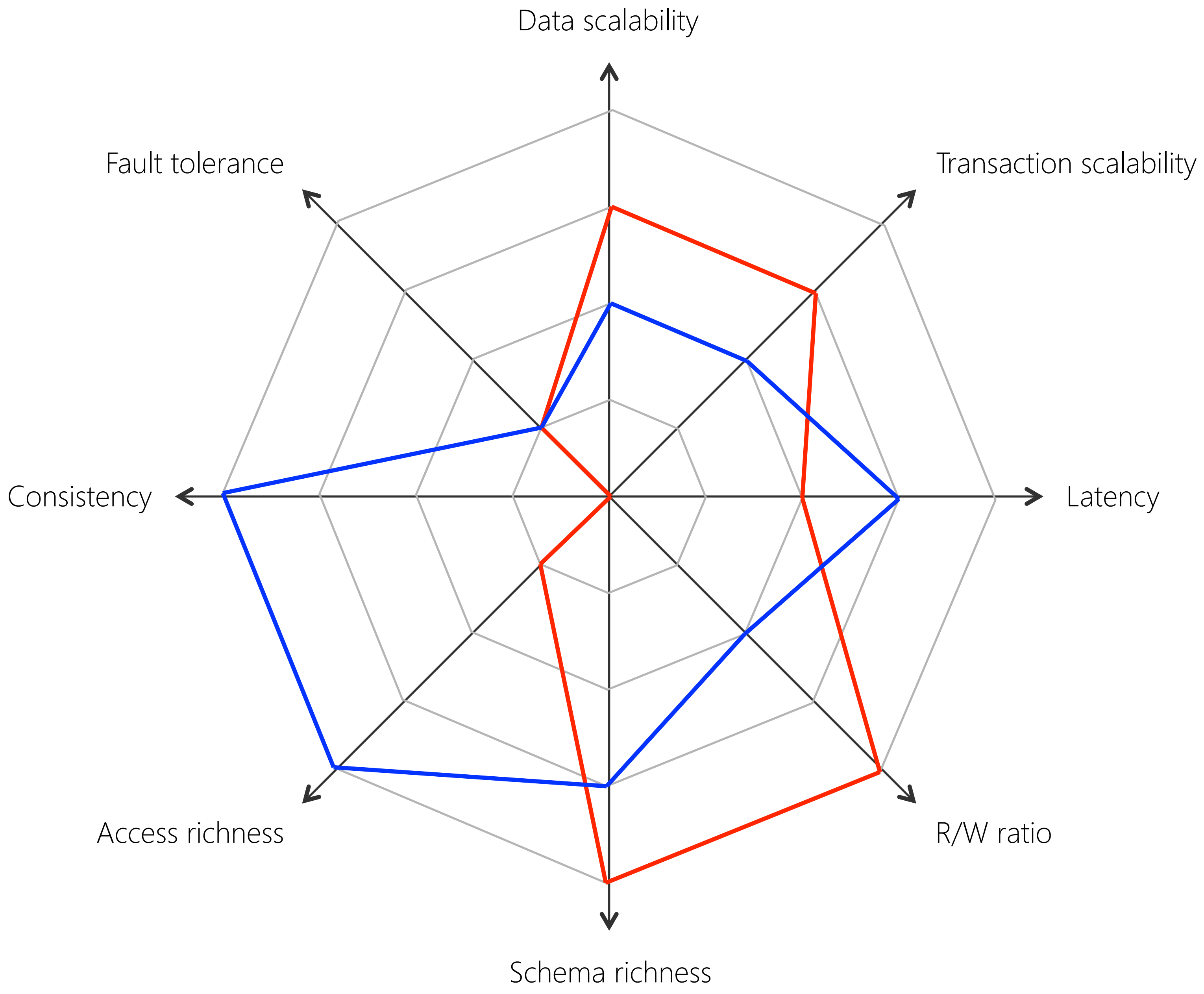
Relational database



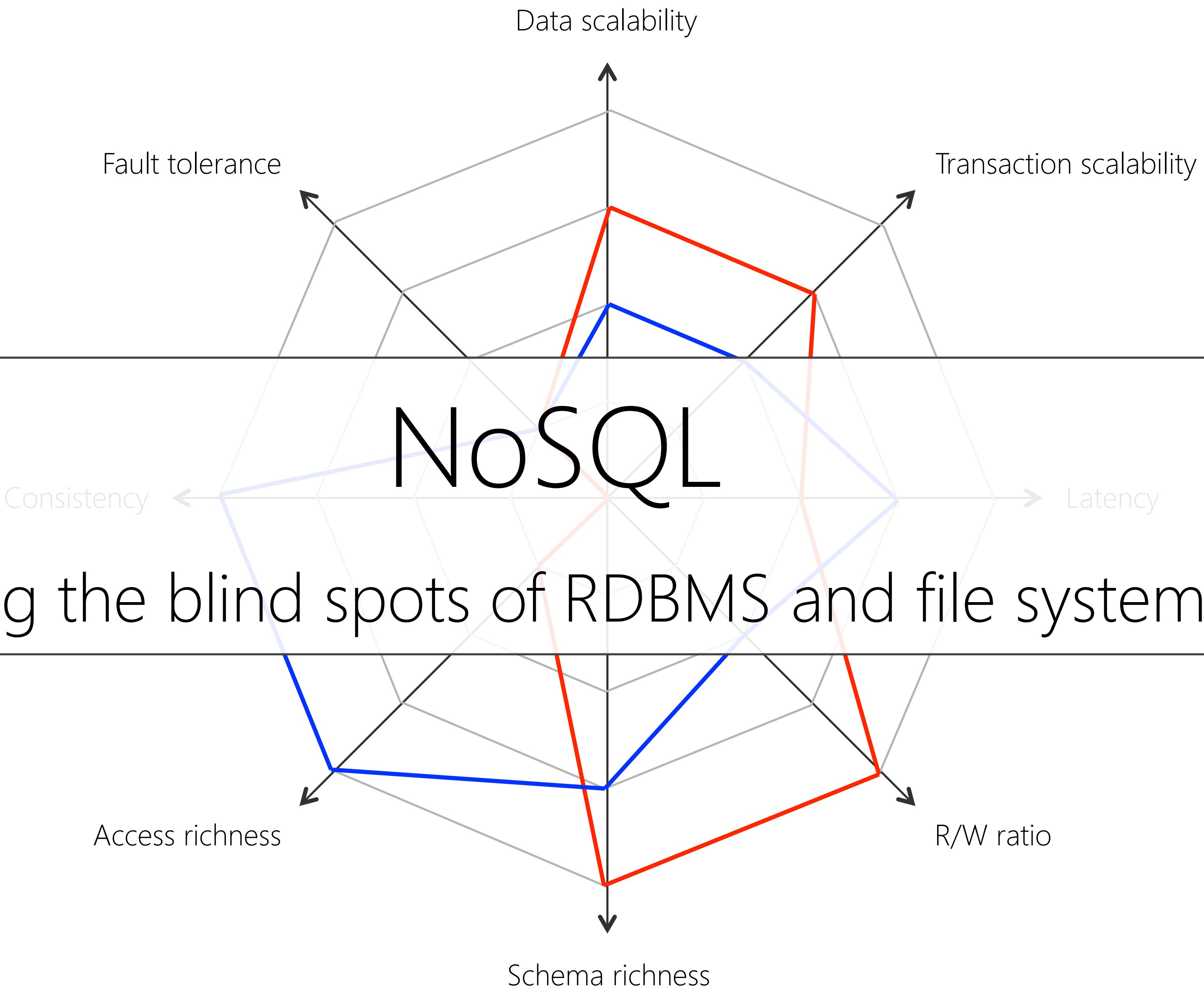
File system


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

RDBMS & file system



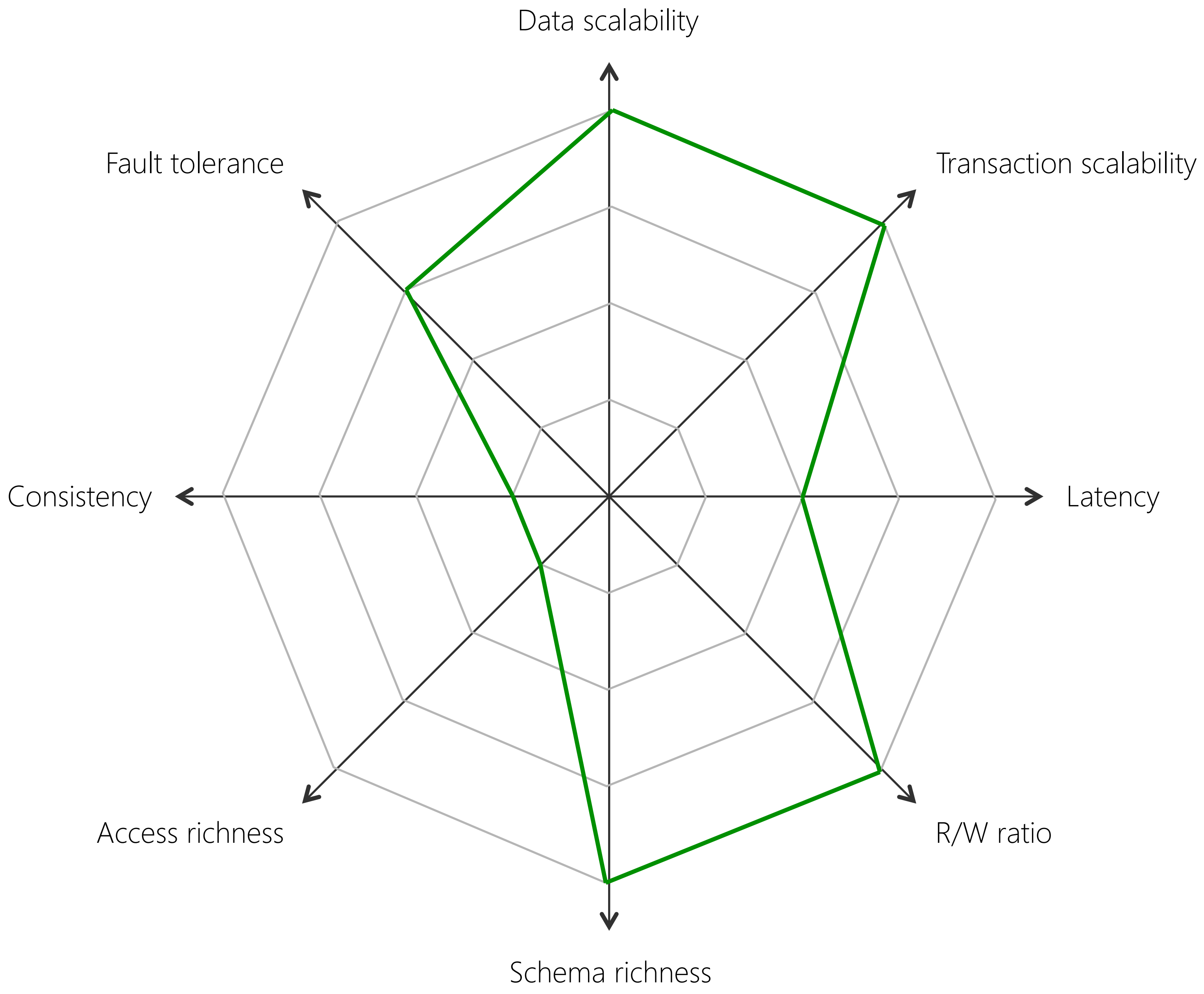
RDBMS & file system



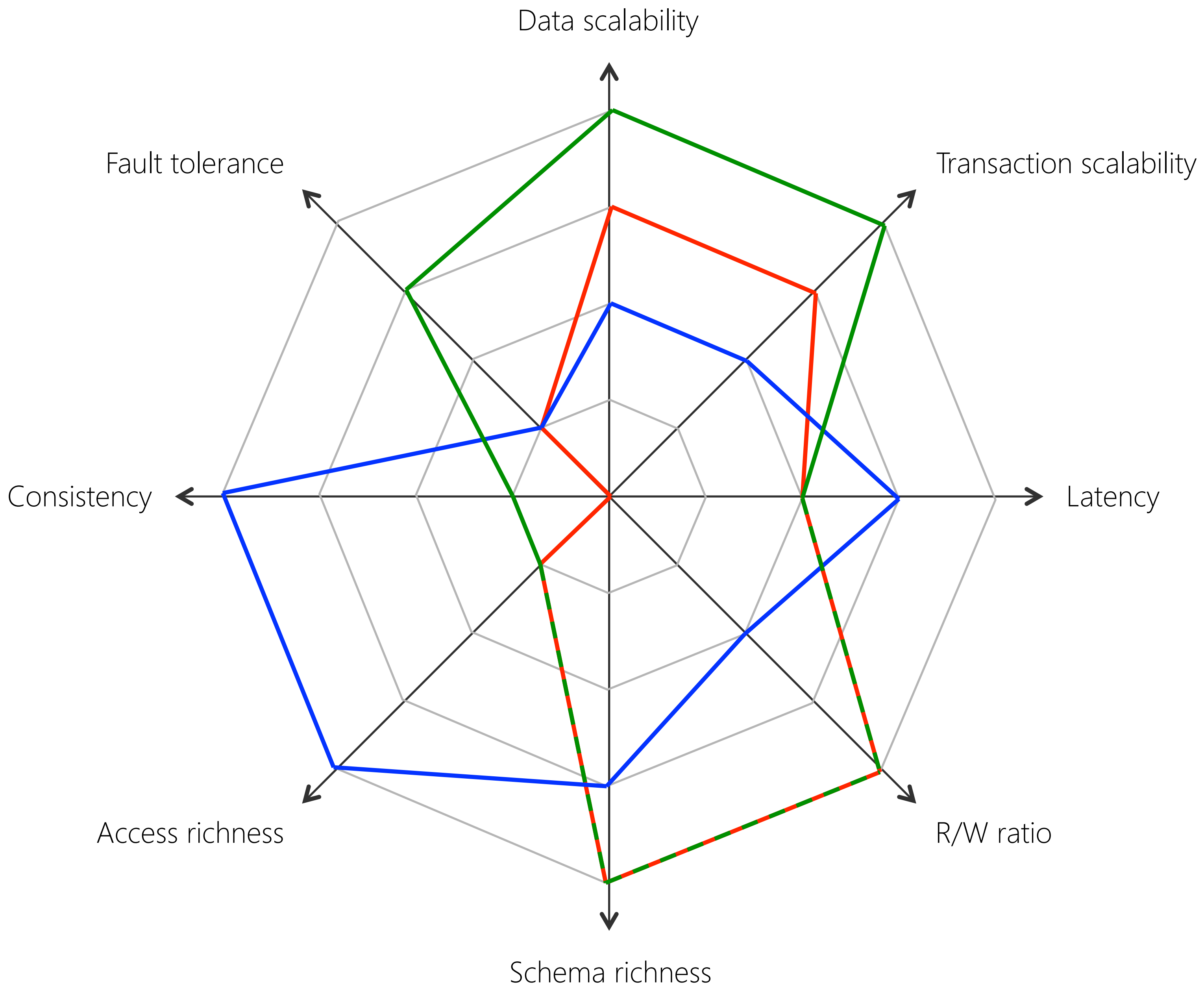
NoSQL

Filling the blind spots of RDBMS and file systems

Cassandra


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

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

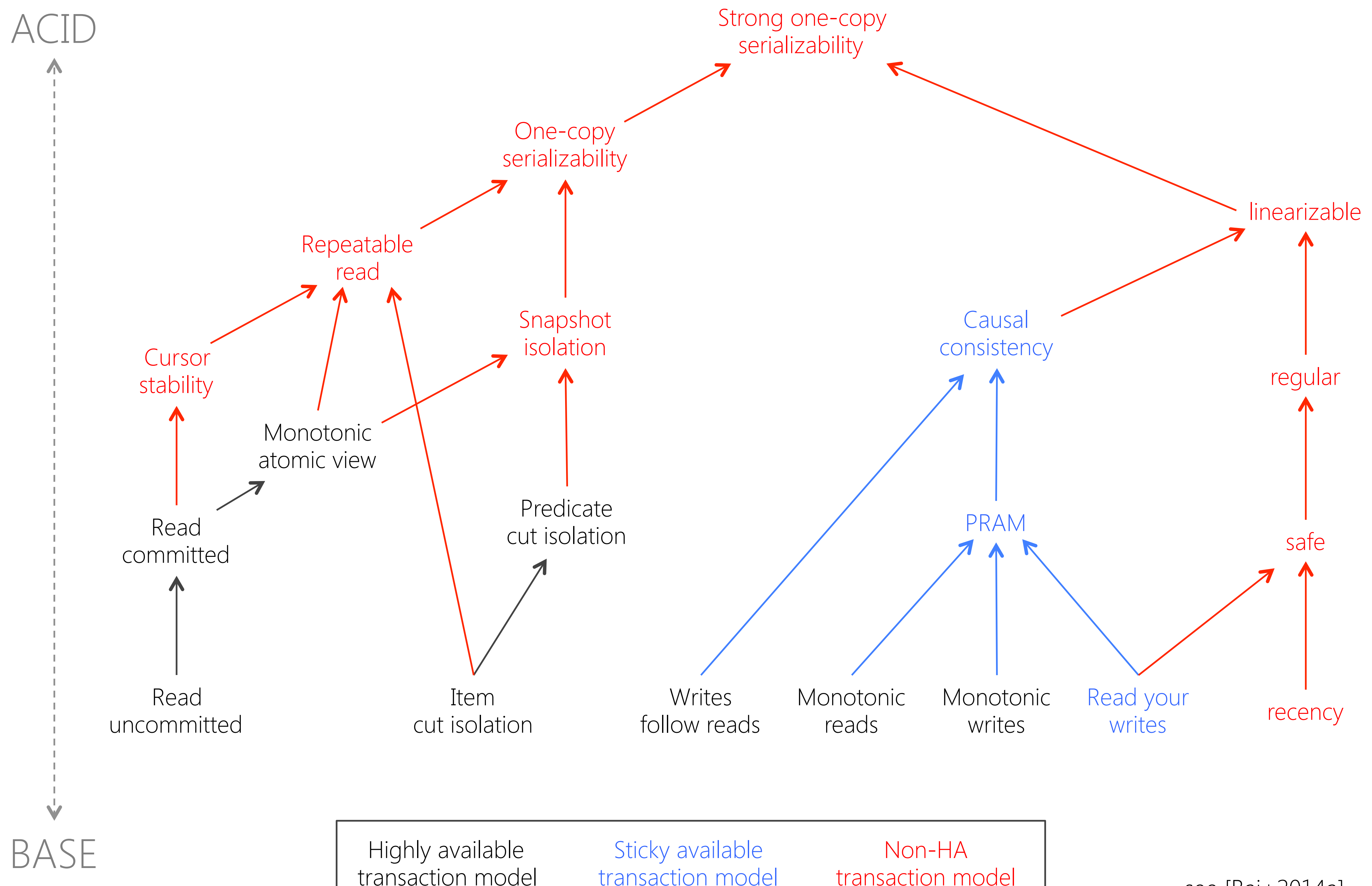
Wrap-up – Present

- IT goes distributed (“scale out”)
- NoSQL fills the empty spots in the storage solution space
- BASE transactions imply a very hard programming model

Future

So, can I only choose between “ACID” and “BASE”?

Remember, that “ACID” does not necessarily mean serializability?



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

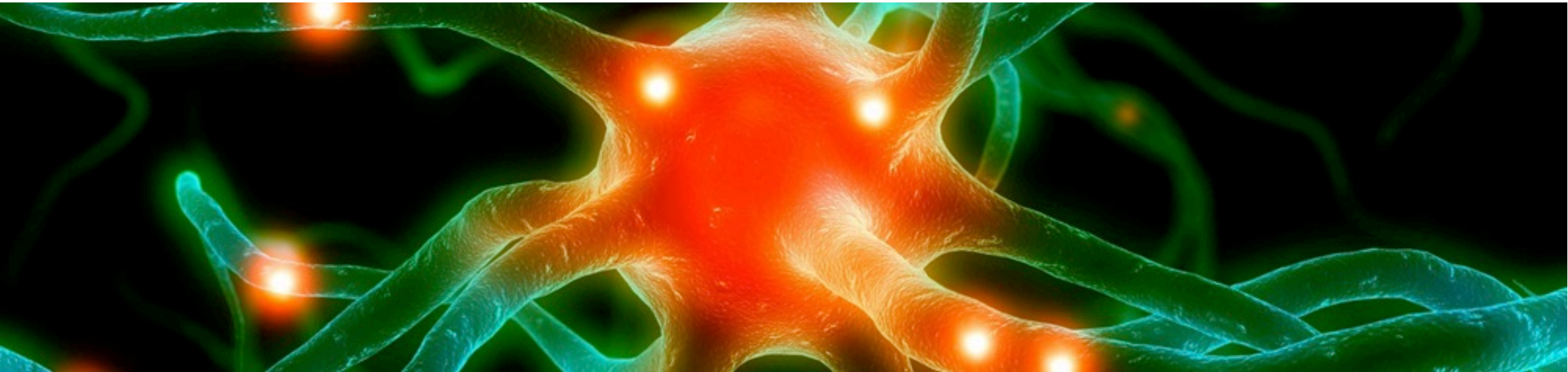
- Lots of options to balance consistency constraints and intricacy of the programming model
- Higher consistency guarantees than "plain BASE" in distributed HA databases may become available in the near future
- Most of them will require effort on the application level

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

[Bai+2013a] Peter Bailis, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, Ion Stoica, "HAT, not CAP: Towards Highly Available Transactions", HotOS 2013

[Bai+2013b] Peter Bailis, Ali Ghodsi, Joseph M. Hellerstein, Ion Stoica, "Bolt-on Causal Consistency", SIGMOD 2013

[Bai+2014a] Peter Bailis, Aaron Davidson, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, Ion Stoica, "Highly Available Transactions: Virtues and Limitations", VLDB 2014

[Bai+2014b] Peter Bailis, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, Ion Stoica, "Scalable Atomic Visibility with RAMP Transactions", SIGMOD 2014

[Bai+2015] Peter Bailis, Alan Fekete, Michael J. Franklin, Ali Ghodsi, Joseph M. Hellerstein, Ion Stoica, "Coordination Avoidance in Database Systems", VLDB 2015

[Bal+2015] Valter Balegas, Sérgio Duarte, Carla Ferreira, Rodrigo Rodrigues, Nuno Preguica, Mahsa Najafzadeh, Marc Shapiro, "Putting Consistency Back into Eventual Consistency", EuroSys 2015

[Ber+1995] Hal Berenson, Phil Bernstein, Jim Gray, Jim Melton, Elizabeth O’Neil, Patrick O'Neil, "A Critique of ANSI SQL Isolation Levels", Microsoft Research, Technical Report MSR-TR-95-51, June 1995

References

[Bre2000] Eric A. Brewer, "Towards Robust Distributed Systems", PODC 2000

[Bur2005] Mark Burgess, "An Approach to Understanding Policy Based on Autonomy and Voluntary Cooperation", DSOM 2005

[Coc2015] Adrian Cockcroft, "Innovation and Architecture", http://de.slideshare.net/adriancockcroft/innovation-and-architecture, S. 122-125

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

[Col2015a-c] Adrian Colyer, "Out of the Fire Swamp", Parts I-III,
http://blog.acolyer.org/2015/09/08/out-of-the-fire-swamp-part-i-the-data-crisis/
http://blog.acolyer.org/2015/09/09/out-of-the-fire-swamp-part-ii-peering-into-the-mist/
http://blog.acolyer.org/2015/09/10/out-of-the-fire-swamp-part-iii-go-with-the-flow/

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

[Cor+2012] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, JJ Furman, et al., "Spanner: Google's Globally-Distributed Database", OSDI 2012

References

[Hel2007] Pat Helland, "Life beyond Distributed Transactions: an Apostate's Opinion", CIDR 2007

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

[Lee+2015] Collin Lee, Seo Jin Park, Ankita Kejriwal, Satoshi Matsushita, John Ousterhout, "Implementing Linearizability at Large Scale and Low Latency", SOSP 2015

[Sha+2011] Marc Shapiro, Nuno Preguiça, Carlos Baquero, Marek Zawirski, "A comprehensive study of Convergent and Commutative Replicated Data Types", Inria Research report, 2011

[Xie+2015] Chao Xie, Chunzhi Su, Cody Littley, Lorenzo Alvisi, Manos Kapritsos, Yang Wang, "High-Performance ACID via Modular Concurrency Control", SOSP 2015

[Zha+2015], Irene Zhang, Naveen Kr. Sharma, Adriana Szekeres, Arvind Krishnamurthy, Dan R. K. Ports, "Building Consistent Transactions with Inconsistent Replication", SOSP 2015

@ufried

Uwe Friedrichsen | uwe.friedrichsen@codecentric.de | https://www.slideshare.net/ufried | https://medium.com/@ufried