GDPRbench

Sam Thomas

Overview of this Discussion

I. Discussing contributions of the paper

II. Motivating the Paper

III. Discussion of GDPRbench compliance model

  A. GDPR primitive operations

  B. DBMS design

IV. Discussion of GDPRbench testing benchmarks

V. Takeaways and Discussion

How well did you feel like you understood the technicalities of GDPRbench?

Contributions?

Contributions

1. GDPR Primitive Operations
2. GDPR Compliant System Design
3. Performance benchmarking

Motivating the Paper

- Databases and data storage practices are the subject of most GDPR compliance fines
- Companies have stopped serving in Europe due to compliance difficulty
- Pioneering database-centered GDPR analysis

Database Compliance

- Establishing primitive GDPR operations and their use frequency
- Proposing database modifications in three types of DBMSs
    - Redis
    - PostgresSQL
    - System C
- Describing resulting overheads and slowdowns

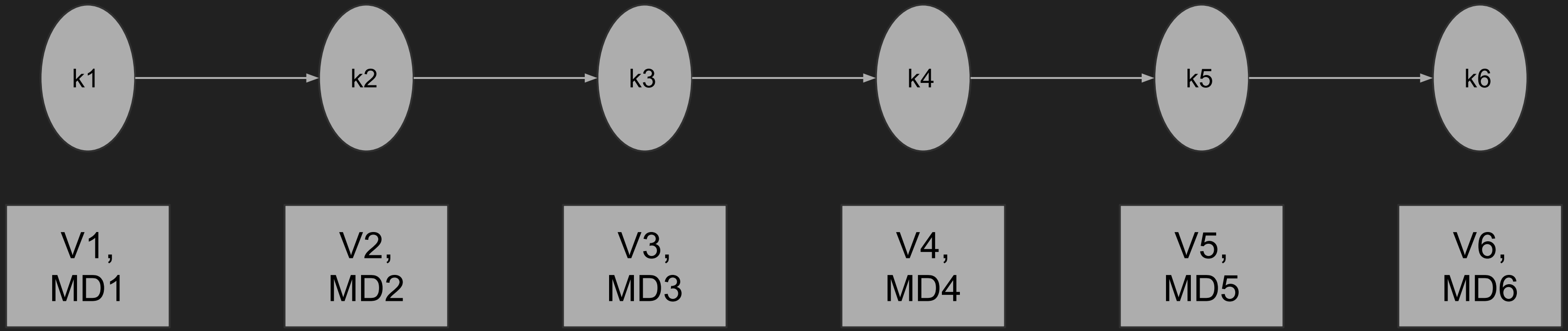
GDPR Primitives

- Mostly read intensive operations with a heavy skew of these operations are from metadata
    - eg. read_metadata_by_key, read_metadata_by_user
- Data- and heuristic-based workloads for each of these primitive operations
    - controller: largely write-intensive operations (add/delete record, update metadata, etc...)
    - customer/data subject: largely read-intensive operations (review data/metadata),
    - regulator: largely read-intensive operations (review system logs, etc…)
    - processor: read-intensive from metadata storage system, normal database protocols for physical data (~20% write, ~80% read)

GDPR-compliant DB Design

- Keep separate database storing metadata and pointers to physical data database
- Add functionality to support previously stated GDPR primitive operations

Redis* (No SQL key-value store)



* note that this is just for demonstration, Redis is likely more like a B+ Tree- , Skip List- , or Hash Table- based architecture that supports some highly parallel operations, but drawing those in PowerPoint wouldn’t be scalable!

PostgresSQL

- Open source, slower SQL implementation
- Many features (some malleable into GDPR-compliant framework)
- System C: similarly SQL-based and feature intensive, optimized and not open source

GDPR Compliant DB Design Takeaways

- None of the storage systems natively support time-to-live (TTL)
- GDPR compliance is less trivial in lightweight storage systems (~400 additional LoC to Redis vs ~100 LoC to PostgresSQL)
- Existing protocols can implement most GDPR necessitated protocols (TLS, etc...)
    - Metadata indexing is especially nontrivial in Redis

Testing Framework

- Based on Yahoo! Cloud Server Benchmark (YCSB)
- Runs tests based on primitive frequency (adjustable knobs)
- Can be used to test other implementations of GDPR compliance

Results - Main Takeaways

- Redis scales the worst in performance and space
- PostgresSQL scales better with lower raw throughput
- “Metadata explosion”
- Performance slowdown up to 5x

Strengths	Weaknesses

Strengths

- System-level analysis of GDPR
- Creates model of GDPR compliance
- Exhaustive list of GDPR compliant primitives and usage frequencies
- Metadata-based additions are non-intrusive adjustments to existing database structures
- Testing benchmarks give good description of performance and spatial overheads of GDPR compliance

Weaknesses

Strengths

- System-level analysis of GDPR
- Creates model of GDPR compliance
- Exhaustive list of GDPR compliant primitives and usage frequencies
- Metadata-based additions are non-intrusive adjustments to existing database structures
- Testing benchmarks give good description of performance and spatial overheads of GDPR compliance

Weaknesses

- Assumes strict reading of GDPR amidst ambiguity
    - Prioritizes system compliance over performance (worst-case overheads)
    - Discussion of GDPR compliance as a spectrum rather than binary
- Presumably other means of satisfying GDPR compliance other than “metadata model”
- GDPR Primitives were highly specific operations, compliance model primitives?

Discussion Questions

- Does GDPRbench measure all possible forms of correctness or one model of correctness?
- Is it fair to assume no information from the database application to optimize performance?
- Do ML-based data scanners like Amazon Macie qualify as sufficient for measuring for GDPR compliance?
- Is designing for GDPR compliance inherently antithetical to reliability, performance and cost?
- All DBMSs considered are row-oriented stores. What about column-oriented stores (Vertica, MemSQL, Apache Kudu) or graph-based?
- Does it make sense to tailor DBMSs for regulators?

Thanks!