ToroDB

Building a NoSQL database on top of a SQL database

@NoSQLonSQL

Álvaro Hernández <*aht@torodb.com*>

About ⟨8K⟩ data

www.8kdata.com

- Research & Development in databases
- Consulting, Training and Support in PostgreSQL
- Founders of PostgreSQL España, 5th largest PUG in the world (>500 members as of today)
- About myself: CTO at 8Kdata:

@ahachete

http://linkd.in/1jhvzQ3

ToroDB in one slide

- Document-oriented, JSON, NoSQL db
- Open source (AGPL)
- MongoDB compatibility (wire protocol level)
- Uses PostgreSQL as a storage backend

Why relational databases: technical perspective

- Document model is very appealing to many. But all dbs started from scratch

- DRY: why not use relational databases? They are proven, durable, concurrent and flexible

- Why not base it on relational databases, like PostgreSQL?

How to map NoSQL (documents) to SQL (relational)

ToroDB storage

- Data is stored in tables. No blobs
- JSON documents are split by hierarchy levels into “subdocuments”, which contain no nested structures. Each subdocument level is stored separately
- Subdocuments are classified by “type”. Each “type” maps to a different table

ToroDB storage (II)

- A "structure" table keeps the subdocument "schema"
- Keys in JSON are mapped to attributes, which retain the original name
- Tables are created dinamically and transparently to match the exact types of the documents

ToroDB storage internals

```
{
    "name": "ToroDB",
    "data": {
        "a": 42, "b": "hello world!"
    },
    "nested": {
        "i": 42,
        "deeper": {
            "a": 21, "b": "hello"
        }
    }
}
```

ToroDB storage internals

The document is split into the following subdocuments:

{ "name": "ToroDB", "data": {}, "nested": {} }

{ "a": 42, "b": "hello world!"}

{ "j": 42, "deeper": {}}

{ "a": 21, "b": "hello"}

ToroDB storage internals

```
select * from demo.t_3
```

did	index	_id	name
0	¤	\x5451a07de7032d23a908576d	ToroDB

```
select * from demo.t_1
```

did	index	a	b
0	¤	42	hello world!
0	1	21	hello

```
select * from demo.t_2
```

did	index	j
0	¤	42

ToroDB storage internals

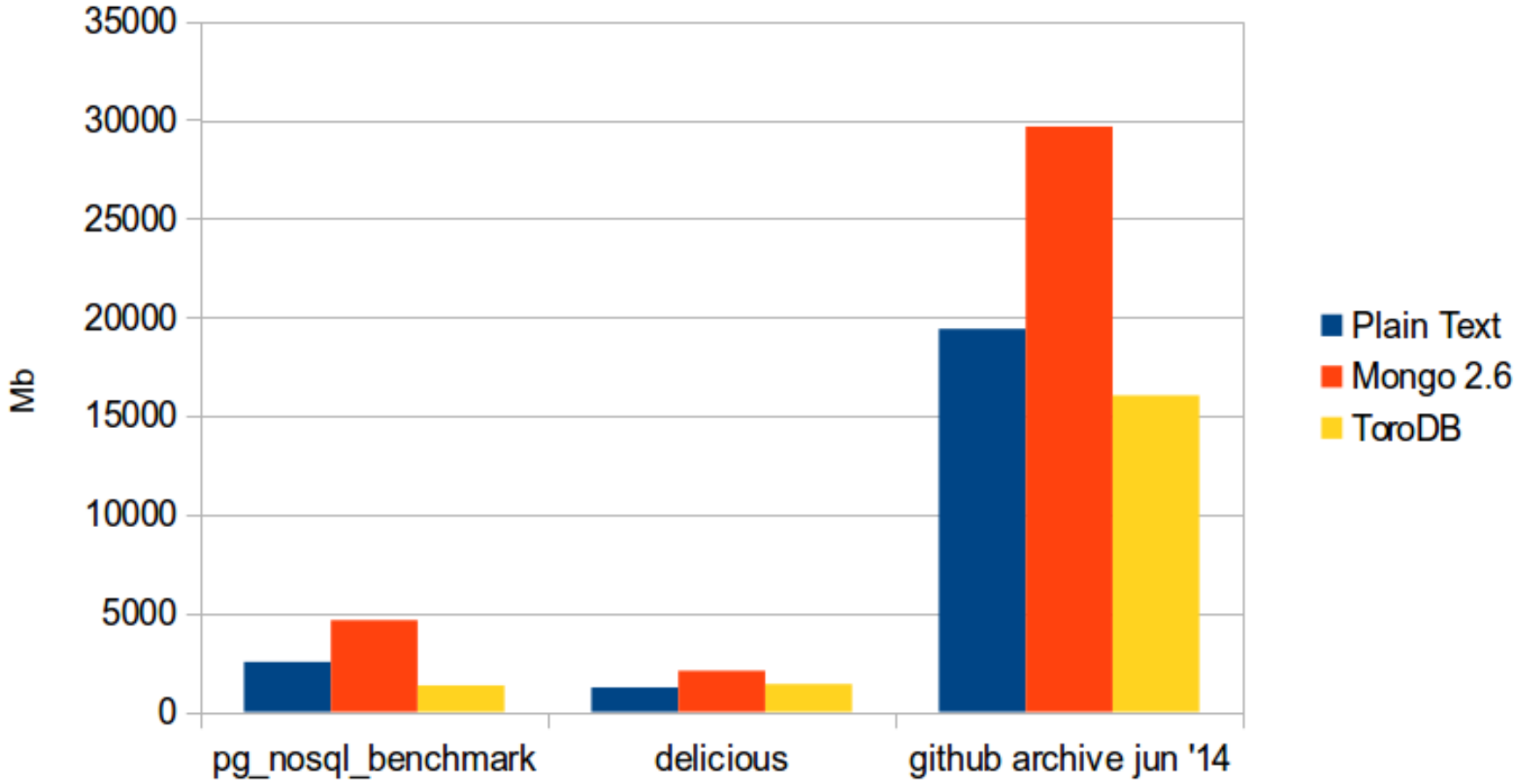
```
select * from demo.structures
```

sid	_structure
0	{"t": 3, "data": {"t": 1}, "nested": {"t": 2, "deeper": {"i": 1, "t": 1}}}

```
select * from demo.root;
```

did	sid
0	0

ToroDB storage and I/O savings



29% - 68% storage required,
compared to Mongo 2.6

ToroDB
The software

The software

ToroDB is written in Java, compatible with versions 6 and above.

- **github.com/torodb**
- RERO policy. Comments, feedback, patches... greatly appreciated
- AGPLv3

ToroDB: Developer Preview

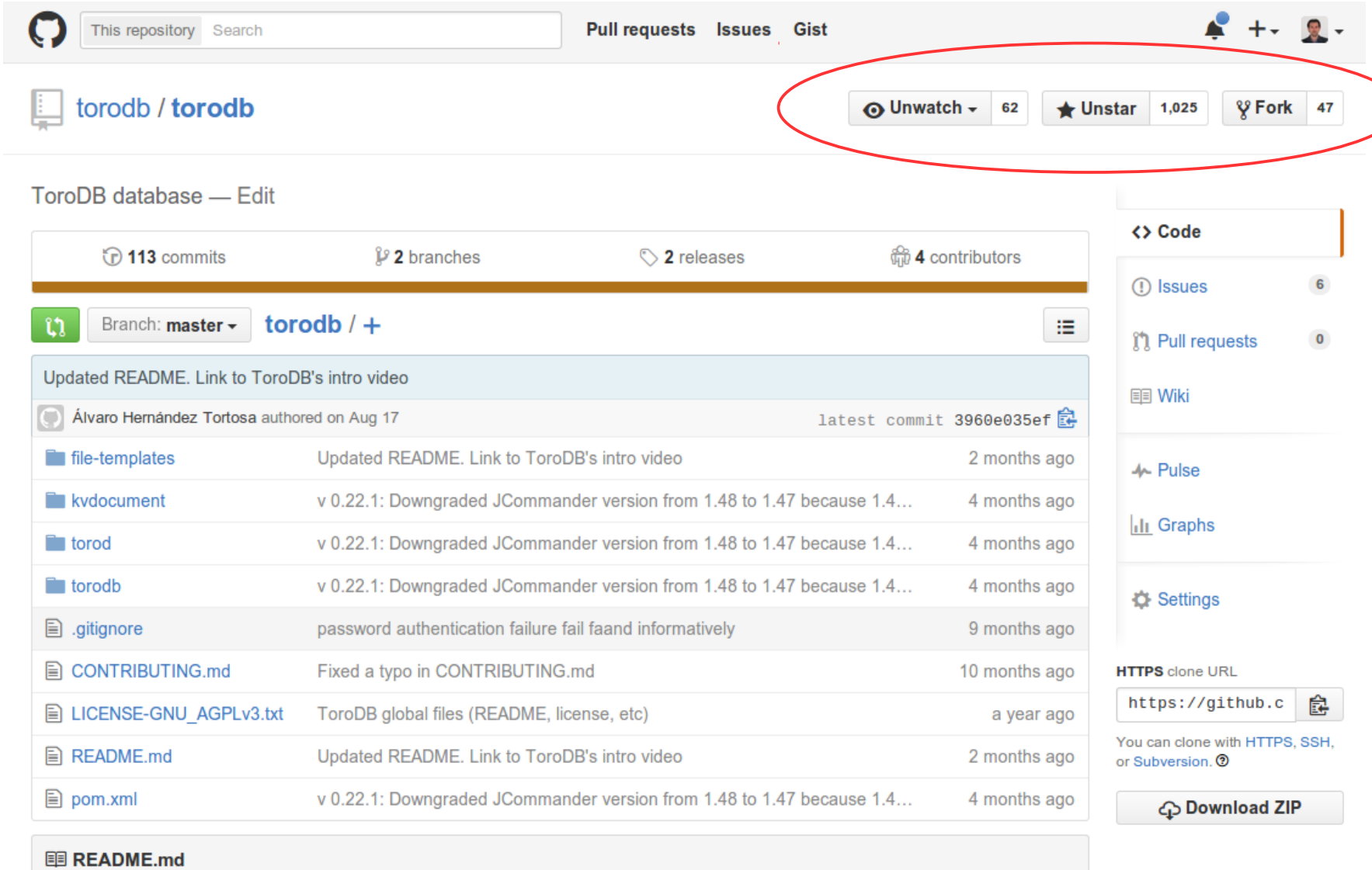
- Clone the repo, build with Maven

- Or download the JAR:
  http://maven.torodb.com/jar/com/torodb/torodb/0.22.1/torodb.jar

- Usage:
  java -jar torodb-*0.22.1*.jar –help
  java -jar torodb-*0.22.1*.jar -d dbname -u dbuser -P 27017

  Connect with normal mongo console!

ToroDB: Community Response



ToroDB: Community Response

Jerónimo López
@jerolba

Following

WOW! ToroDb es trending en GitHub! github.com/trending Une lo mejor de MongoDb con lo mejor de las BD de verdad! github.com/torodb/torodb

@nelhage
@nelhage

Follow

As nice as SQL storage engines are these days, projects like ToroDB speak to a real problem: SQL is a fucking terrible API.

Lukas Eder
@lukaseder

Follow

Wow, kick-ass start for ToroDB after Hackernews link! github.com/torodb/torodb

Charity Majors
@mipsytipsy

Follow

more proof that everyone wants mongodb's data interface layer with better way of laying bits on disk. github.com/torodb/torodb via @nelhage

Gunboat Laundromat
@peschkaj

Follow

@cheeseplus Why not ToroDB? github.com/torodb/torodb It's like Mongo, but it's not built on a platform of fail!

Alvaro Hernandez T
@ahachete

ToroDB (github.com/torodb/torodb) is #2 in trending repos today in github!!! github.com/trending Star it!

Neil Brennan, Esq.
@nellophonic

Follow

ToroDB. Speaks natively the MongoDB protocol AND saves your data. github.com/torodb/torodb

Andrew Martin
@sublimino

Follow

Loving the look of github.com/torodb/torodb - speaks Mongo protocol with a Postgres relational backend. Best of both? #nosql

Going beyond NoSQL

Going beyond NoSQL

MongoDB popularized the document model that many love.

But can we go further than that?

Can't the foundation of relational databases provide a basis for offering new features on a NoSQL, document-like, JSON database?

The schema-less fallacy

```
{
  “name”: “Álvaro”,
  “surname”: “Hernández”,
  “height”: 200,
  “hobbies”: [
    “PostgreSQL”, “triathlon”
  ]
}
```

The schema-less fallacy

```
{
  "name": "Álvaro",
  "surname": "Hernández",
  "height": 200,
  "hobbies": [
    "PostgreSQL", "triathlon"
  ]
}
```

metadata → Isn't that... **schema**?

The schema-less fallacy: BSON

```
{
  “name”: (string) “Álvaro”,
  “surname”: (string) “Hernández”,
  “height”: (number) 200,
  “hobbies”: {
        “0”: (string) “PostgreSQL” ,
        “1”: (string) “triathlon”
  }
}
```

metadata → Isn't that... **schema**?

The schema-less fallacy

- It's not schema-less
- It is *"attached-schema"*
- It carries an overhead which is not 0

Schema-attached repetition

```
{ "a": 1, "b": 2 }
{ "a": 3 }
{ "a": 4, "c": 5 }
{ "a": 6, "b": 7 }
{ "b": 8 }
{ "a": 9, "b": 10 }
{ "a": 11, "b": 12, "¡": 13 }
{ "a": 14, "c": 15 }
```

Counting "document types" in collections of millions: at most, 1000s of different types

Schema-attached repetition

How data is stored in schema-less

This is how we store in ToroDB

ToroDB: query "by structure"

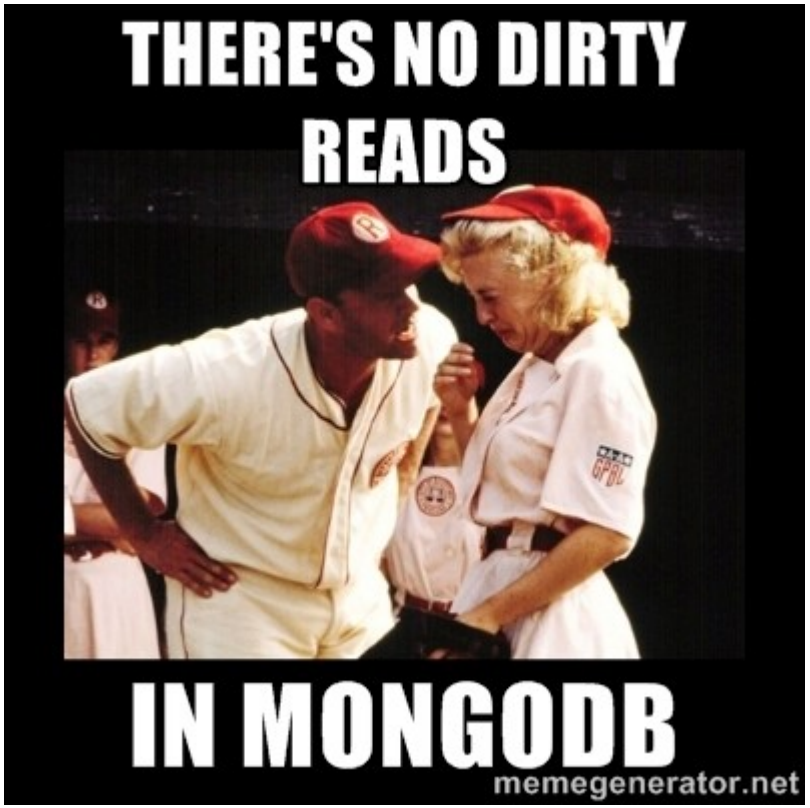
- ToroDB is effectively **partitioning by type**
- Structures (schemas, partitioning types) are cached in ToroDB memory
- Queries only scan a subset of the data
- Negative queries are served directly from memory

Cheap single-node durability

- Without journaling, MongoDB is not durable nor crash-safe

- MongoDB requires "**j: true**" for true single-node durability. But who guarantees its consistent usage? Who uses it by default?
  j:true creates I/O storms equivalent to SQL CHECKPOINTs

“Clean” reads



Oh really?

“Clean” reads

http://docs.mongodb.org/manual/reference/write-concern/#read-isolation-behavior

“MongoDB will allow clients to read the results of a write operation before the write operation returns.”

“If the mongod terminates before the journal commits, even if a write returns successfully, queries may have read data that will not exist after the mongod restarts.”

“Other database systems refer to these isolation semantics as read uncommitted.”

"Clean" reads

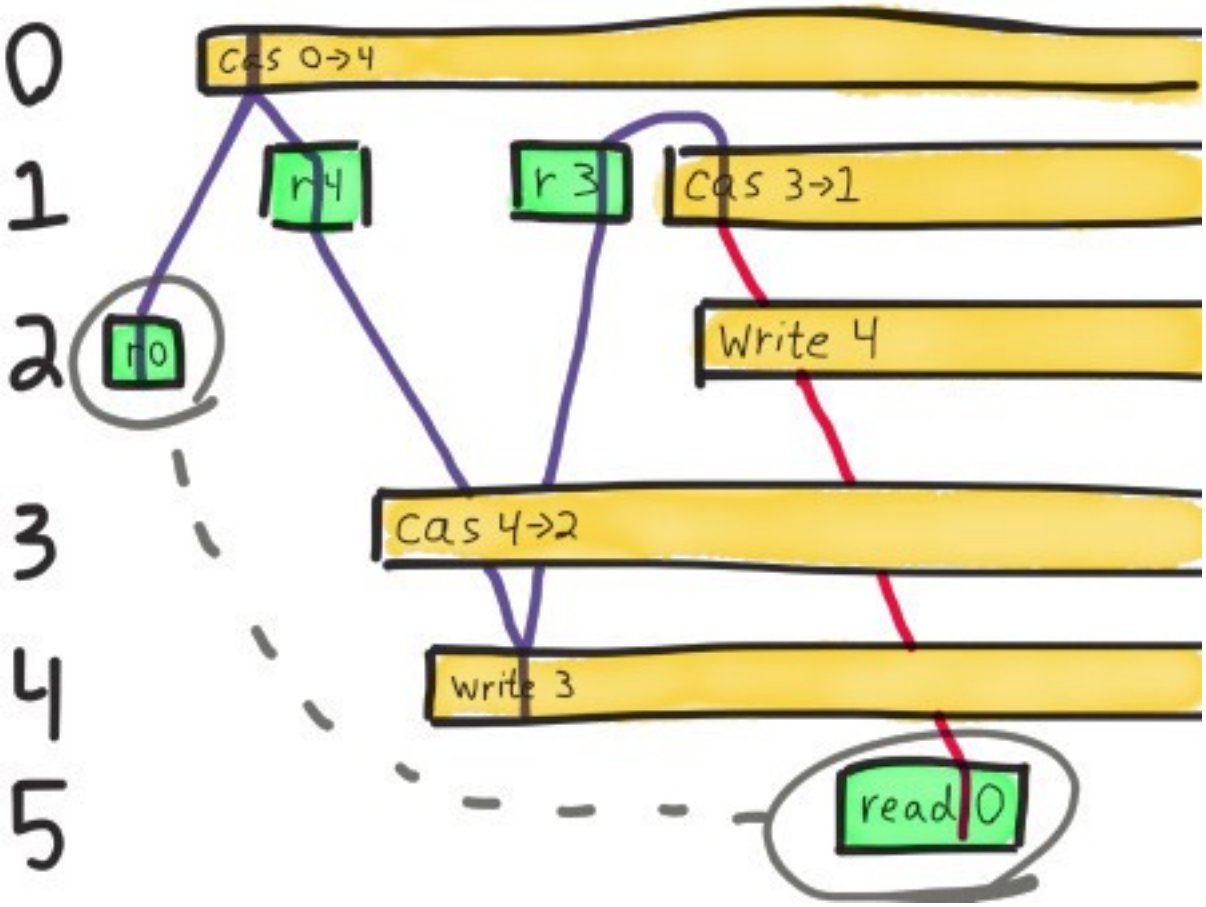
Thus, MongoDB suffers from **dirty reads**. Or probably better called "**tainted reads**".

What about $snapshot? Nope:

"The snapshot() does not guarantee that the data returned by the query will reflect a single moment in time nor does it provide isolation from insert or delete operations."

http://docs.mongodb.org/manual/faq/developers/#faq-developers-isolate-cursors

“Clean” reads
“Call me maybe”



https://aphyr.com/posts/322-call-me-maybe-mongodb-stale-reads

ToroDB: going beyond MongoDB

- Cheap single-node durability
  PostgreSQL is 100% durable. Always. And it's cheap (doesn't do I/O storms)

- "Clean" reads
  Cursors in ToroDB run in repeatable read, read-only mode:

```
globalCursorDataSource.setTransactionIsolation("TRANSACTIO
N_REPEATABLE_READ");
globalCursorDataSource.setReadOnly(true);
```

Atomic operations

- There is no support for atomic bulk insert/update/delete operations

- Not even with $isolated:

"Prevents a write operation that affects multiple documents from yielding to other reads or writes [...] You can ensure that no client sees the changes until the operation completes or errors out. **The $isolated isolation operator does not provide "all-or-nothing" atomicity for write operation**s."

http://docs.mongodb.org/manual/reference/operator/update/isolated/

High concurrency

- MMAPv1 is still collection-locked
- WiredTiger is document-locked
- But still exclusive locks (MMAP). Most relational databases have MVCC, which means almost conflict-free readers and writers at the same time

ToroDB: going beyond MongoDB

- Atomic bulk operations

By default, bulk operations in ToroDB are atomic. Use flag ContinueOnError: 1 *to perform non-atomic bulk operations*

- Highest concurrency

PostgreSQL uses MVCC. Readers and writers do not block each other. Writers block writers only for the same record

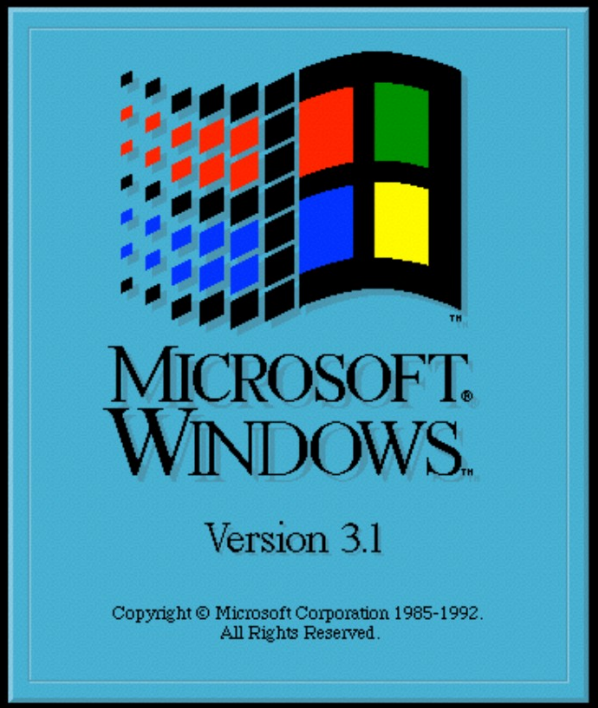
ToroDB: native SQL

- NoSQL is trying to get back to SQL
- ToroDB is SQL **native**!
- Insert with Mongo, query with SQL!
- Powerful PostgreSQL SQL: window functions, recursive queries, hypothetical aggregates, lateral joins, CTEs, etc

ToroDB: native SQL

Still using
Windows 3.1?

So why stick to
SQL-92?



@ModernSQL in PostgreSQL
@MarkusWinand

ToroDB
Experimental. The future

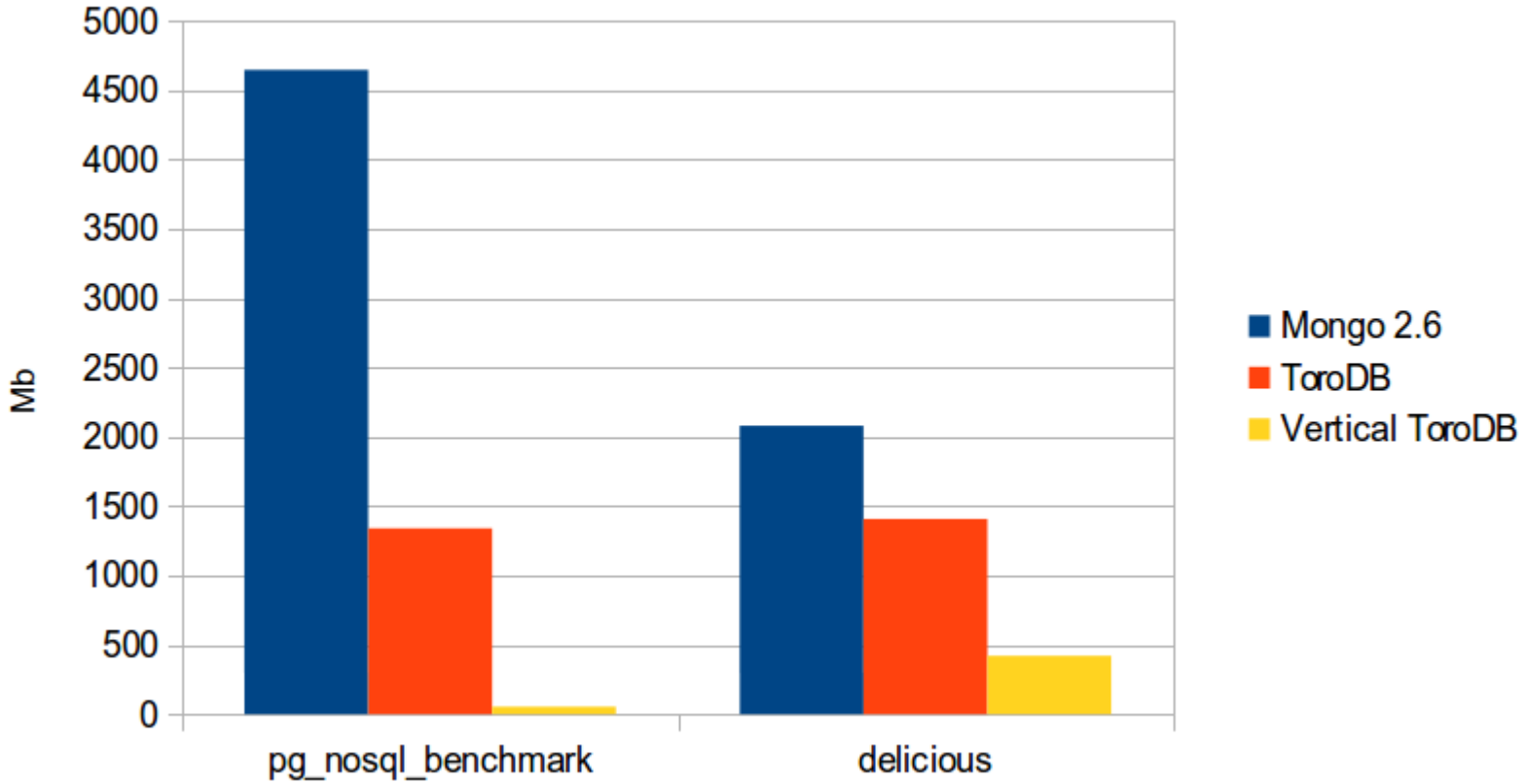
ToroDB: Experimental research directions

- User columnar storage (CitusDB)
- Use Postgres-XL as a backend. This requires us to distribute ToroDB's cache (ehcache, Hazelcast)
- Use pg_shard for sharding

Big Data speaking mongo: Vertical ToroDB

What if we use CitusData's cstore to store the JSON documents?

Big Data speaking mongo: Vertical ToroDB



1.17% - 20.26% storage required, compared to Mongo 2.6


8K
data