Tarantool - a NoSQL database with SQL

Pavel Lapaev, Product Manager@Mail.ru
Kirill Yukhin, Engineering Manager @Mail.ru
Agenda

- What is Mail.ru Group?
- What is Tarantool?
- Performance
- Storage engines
- Scaling
- Why SQL?
- Roadmap
Mail.ru Group

20 years in business, leading IT company in Russia

- Social networks VK (97m monthly) and Odnoklassniki (45m monthly)
- Email (top 5 in the world, 100m active accounts)
- Portal and IM (35m monthly)
- Online Games (512m accounts)
- E-commerce, Search, Delivery, Marketplace, E-learning, Maps, etc.
Tarantool in a Nutshell

An in-memory database with an integrated application server

- Team of 70+ people
- 10 years of history
- Open-source and enterprise versions
Tarantool Facts

Here is a bunch of features:

- In-memory and disk storage engines
- Core written in C, app server exposes Lua
- Persistence (WAL and snapshots)
- Application server onboard
- ACID transactions
- Horizontal scalability: sharding and replication
- NoSQL... with SQL
Tarantool Products

- Tarantool itself
- Cartridge (cluster management framework)
- Kubernetes Operator

- Enterprise Edition
- Data Grid
Enterprise Products

- Enterprise Edition
  - L2, L3 support
  - Enterprise database connectivity
  - Oracle replication modules
  - Security audit log
- Data Grid
  - System to develop distributed apps
  - Flexible connectivity to external sources
  - Versioned data storage
  - Pre and post processing of data
  - Lots of tools already in the box
Tarantool Customers

Avito  badoo  Alfa·Bank
mastercard  Yota  Beeline
Wargaming  Мегафон  Газпромнефть
History

- Created @ Mail.ru Group about 10 years ago
- Used to store sessions/profiles of millions of users

Web servers

- load web-page
- AJAX request
- mobile API

> 1.000.000 requests per second

4 instances

profiles

8 instances
Must-have and mustn't-have features

- No secondary keys, constraints etc.
- Schema-less
- Need a language. *QL is not* must-have
  - High-speed in any sense!
  - Simple
  - Extensible
- Transactions
- Persistency
- Once again: it must be **fast**, no excuses
Tarantool: Bird's Eye View

- No need for cache: It is **in-memory**
- But still DBMS: persistency and transactions
  - It regards **ACID**
- Single threaded: It is **lock-free**
- Easy: imperative language is on board: Lua
  - It **JITs**
    - It's easy to program for business
- It scales: **Replication** and sharding
DBMS + Application Server
C, Lua, SQL, Python, PHP, Go, Java, C# ...
Persistent in-memory and disk storage engines
Stored procedures in C, Lua, SQL
Coöperative multitasking

Multithreading

Fibers

Event-loop
Coöperative multitasking

**Multithreading**

- Losses on caches coherency support
- Losses on locks
- Losses on long operations

That is a **stall**

---

**Fibers**

---

**Event-loop**
Coöperative multitasking

Multithreading

That is a stall

- Losses on caches coherency support
- Losses on locks
- Losses on long operations

Fibers

Event-loop

- Thread is always busy
- Lock-free
- Single core - no coherency issues at all
Vinyl

• In-memory is OK, but not always enough
• Write-oriented: LSM tree
• Same API as memtx
• Transactions, secondary keys

<table>
<thead>
<tr>
<th></th>
<th>Tarantool/Memtx</th>
<th>Tarantool/Vinyl</th>
<th>MySQL (InnoDB), Oracle, Postgres</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read workload</td>
<td>Heavily optimized</td>
<td>Just normal</td>
<td>Just normal</td>
</tr>
<tr>
<td>Write workload</td>
<td>Heavily optimized</td>
<td>Heavily optimized</td>
<td>Just normal</td>
</tr>
<tr>
<td>Dataset limit</td>
<td>RAM</td>
<td>RAM x 100</td>
<td>?</td>
</tr>
</tbody>
</table>
Scaling

Why?
Scaling

Why?
Scaling

Vertical
Scaling

Horizontal
Horizontal scaling

Replication

ABC  ABC  ABC
Scaling computation and fault tolerance

Sharding

A  C  B
Scaling computation and data
Horizontal scaling

Replication
ABC ABC ABC
Scaling computation and fault tolerance

Sharding
A B C
Scaling computation and data

Replication and sharding

Scaling computation, data and fault tolerance
Replication

Asynchronous

begin → replicate
replicate → commit

Synchronous

begin → prepare
prepare → replicate
replicate → commit
Replication

Asynchronous

begin → commit

replicate

Commit is not waiting for replication to succeed

Synchronous

begin → prepare

commit → replicate
Replication

Asynchronous

\[ \text{begin} \rightarrow \text{commit} \rightarrow \text{replicate} \]

Commit is not waiting for replication to succeed

Synchronous

\[ \text{begin} \rightarrow \text{prepare} \rightarrow \text{replicate} \rightarrow \text{commit} \]

Two phase commit. To succeed, need to replicate to N nodes
Replication

Asynchronous

```
begin → commit
replicate
```

Commit is not waiting for replication to succeed

- Faster
- Replicas might lag, conflict

Synchronous

```
begin → prepare → replicate
commit
```

Two phase commit. To succeed, need to replicate to N nodes
Replication

**Asynchronous**

- begin
- commit
- replicate

Commit is not waiting for replication to succeed

- Faster
- Replicas might lag, conflict

**Synchronous**

- begin
- prepare
- replicate
- commit

Two phase commit. To succeed, need to replicate to N nodes

- More reliable
- Slower, complicated protocols
Sharding

Ranges | Decide where to store? | hash

min ───► max

Found range where the key belongs -> found the node

Calculated hash of the key -> found the node
Sharding

Ranges | Decide where to store? | hash

min ─────────────── max

Found range where the key belongs -> found the node

Calculated hash of the key -> found the node

Cockroach DB

mongoDB
Sharding

Ranges  Decide where to store?  hash

min  \rightarrow max

Found range where the key belongs -> found the node

Cockroach DB

mongoDB

• Best
• Complicated
• Usually useless

Calculated hash of the key -> found the node
Sharding

Ranges Decide where to store? hash

min max

Found range where the key belongs -> found the node

Cockroach DB

mongoDB

• Best
• Complicated
• Usually useless

Calculated hash of the key -> found the node

TARANTOOL
Sharding

- Complex queries not fast
- Complex resharding
- Complicated
- Usually useless
- Good enough

Found range where the key belongs ->

Calcualted hash of the key ->

Found the node

TARANTOOL

Cockroach DB

Mongo DB

Best

Complex resharding

Hash

Ranges

Decide where to store?
Resharding problem

\[
\text{shard\_id}(\text{key}) : \text{key} \rightarrow \{\text{shard}_1, \text{shard}_2, \ldots, \text{shard}_N\}
\]

Change N leads to change of shard-function

\[
\text{shard\_id}(\text{key}_1) \neq \text{new\_shard\_id}(\text{key})
\]
Resharding problem

\[ \text{shard\_id}(\text{key}) : \text{key} \rightarrow \{ \text{shard}_1, \text{shard}_2, \ldots, \text{shard}_N \} \]

Change N leads to change of shard-function

\[ \text{shard\_id}(\text{key}_1) \neq \text{new\_shard\_id}(\text{key}) \]

- Need to re-calculate shard-functions for all data
- Some data might move on one of old nodes

Useless data moves
Resharding problem

\[ \text{shard\_id(key)} : \text{key} \rightarrow \{\text{shard}_1, \text{shard}_2, ..., \text{shard}_N\} \]

Change \( N \) leads to change of shard-function

\[ \text{shard\_id(key1)} \neq \text{new\_shard\_id(key)} \]

- Need to re-calculate shard-functions for all data
- Some data might move on one of old nodes

... but not in Tarantool land
Virtual sharding

Data
{tuple}

Virtual nodes

Physical nodes
Virtual sharding

Data
{tuple}

Virtual nodes

Physical nodes

\[ \text{shard}_id(\text{key}) = \{\text{bucket}_1, \text{bucket}_2, \ldots, \text{bucket}_N\} \]

\# \text{\ mini-bucket} = \text{const} >> \# \text{\ db node}

Shard-function is \textbf{fixed}
Virtual sharding

Data

{tuple}

Virtual nodes

{tuple} {tuple} {tuple}

{tuple} {tuple}

{tuple}

Physical nodes

shard_id(key) = \{bucket_1, bucket_2, ..., bucket_N\}

# bucket = const >> # node

Shard-function is fixed

TARANTOOL

Couchbase
Sharding

- Ranges
- Hashes
- Virtual buckets

Having a range or a bucket, how to find where it is stored physically?
Sharding

- Ranges
- Hashes
- Virtual buckets

Having a range or a bucket, how to find where it is stored physically?

1. Prohibit re-sharding
Sharding

- Ranges
- Hashes
- Virtual buckets

Having a range or a bucket, how to find where it is stored physically?

1. Prohibit re-sharding
2. Always visit all nodes
Sharding

- Ranges
- Hashes
- Virtual buckets

Having a range or a bucket, how to find where it is stored physically?

1. Prohibit re-sharding
2. Always visit all nodes
3. Implement proxy-router!
Why SQL?

CREATE TABLE t1 (id INTEGER PRIMARY KEY, a INTEGER, b INTEGER, c INTEGER)
CREATE TABLE t2 (id INTEGER PRIMARY KEY, x INTEGER, y INTEGER, z INTEGER)

SQL> SELECT DISTINCT(a)
    FROM t1, t2
    WHERE t1.id = t2.id
    AND t2.y > 1;
Why SQL?

CREATE TABLE t1 (id INTEGER PRIMARY KEY, a INTEGER, b INTEGER, c INTEGER)
CREATE TABLE t2 (id INTEGER PRIMARY KEY, x INTEGER, y INTEGER, z INTEGER)

function query()
    local join = {}
    for _, v1 in box.space.t1:pairs({}, {iterator='ALL'}) do
        local v2 = box.space.t2:get(v1[1])
        if v2[3] > 1 then
            table.insert(join, {t1=v1, t2=v2})
        end
    end
    local dist = {}
    for _, v in pairs(join) do
        if dist[v['t1'][2]] == nil then
            dist[v['t1'][2]] = 1
        end
    end
    local result = {}
    for k, _ in pairs(dist) do
        table.insert(result, k)
    end
    return result end
SQL Features

- Trying to be subset of ANSI
- Minimum overhead of query planner
- ACID transactions, SAVEPOINTs
- left/inner/natural JOIN, UNION/EXCEPT, subqueries
- HAVING, GROUP BY, ORDER BY
- WITH RECURSIVE
- Triggers
- Views
- Constraints
- Collations
Perspectives

- Onboard sharding
- Synchronous replication
- SQL: more types, JIT, query planner
<table>
<thead>
<tr>
<th>Feature</th>
<th>Tarantool VShard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sharding</td>
<td>✔</td>
</tr>
<tr>
<td>Replication</td>
<td>✔</td>
</tr>
<tr>
<td>In-memory</td>
<td>✔</td>
</tr>
<tr>
<td>Disk</td>
<td>✔</td>
</tr>
<tr>
<td>Persistency</td>
<td>✔</td>
</tr>
<tr>
<td>SQL</td>
<td>✔</td>
</tr>
<tr>
<td>Stored procedures</td>
<td>✔</td>
</tr>
<tr>
<td>Audit logging</td>
<td>✔</td>
</tr>
<tr>
<td>Connectors to DBMSes</td>
<td>✔</td>
</tr>
<tr>
<td>Static build</td>
<td>✔</td>
</tr>
<tr>
<td>GUI</td>
<td>✔</td>
</tr>
<tr>
<td>Unprecedented performance</td>
<td>✔</td>
</tr>
</tbody>
</table>

- **Memtx engine**
- **Vinyl engine, LSM-tree**
- **Both engines**
- **ANSI subset**
- **Lua, C, SQL**
- **Yes**
- **MySQL, Oracle, Memcached**
- **for Linux**
- **Cluster management**
- **100.000 RPS per instance - easy!**
Why do we need Tarantool at Enterprise?

Oleg Ivlev
Head of Digital Service Architecture Office @ MegaFon
What Enterprises Want?

1. **Time To Market**
   - Better time to market than in the Industry and speed as enabler for the Partners

2. **Quality**
   - Outstanding customer experience and advanced customer care in the digital age

3. **TCO**
   - Total cost of ownership under control and manageable growth of business enablers
The 3-speed of IT

**Front End layer principle**
- Mainly agile
- Fast TTM (daily changes)
- No business logic (presentation)

**Middle layer principle**
- Reusability
- 80/20 rules agile
- Medium TTM (weekly changes)
- Internet scale high availability (SSO, caching, fault tolerance)
- Business customer logic host
- Multi-vendor open ecosystem

**Back End layer principle**
- Focus on core capabilities of the factory (platforms)
- Mainly waterfall => long TTM
- Factory business logic host
Digital Ecosystem at MegaFon
Evolution of caches for Real Time apps

Application specific caches in C

- New specific cache for new Real Time application
- No replication of data between caches in different Data Centers
- Disaster Recovery procedure uses manual switch to standby DB
Evolution of caches for Real Time apps

In-Memory DB Cluster for distributed RT application

- In-memory DB as distributed cache for RT application
- Out of the box support of Disaster Recovery in multiple Data Centers
- Cross Data Center synchronisation
- Messaging instead of file transfer
Tarantool roles at Enterprise

**Cache**
- External cache for DB of records
- Internal cache of a platform
- Transactional cache

**In memory DB**
- In Memory DB
- Microservice
- Data Grid

**Event processing**
- Online replication
- Change Data Capture
- Complex Event Processing
Спасибо!

https://tarantool.io
https://github.com/tarantool/tarantool