Orioledb and a real use-case from Arsenal Platform

Aliaksei Ramanau,
Engineering Team Lead, Wargaming

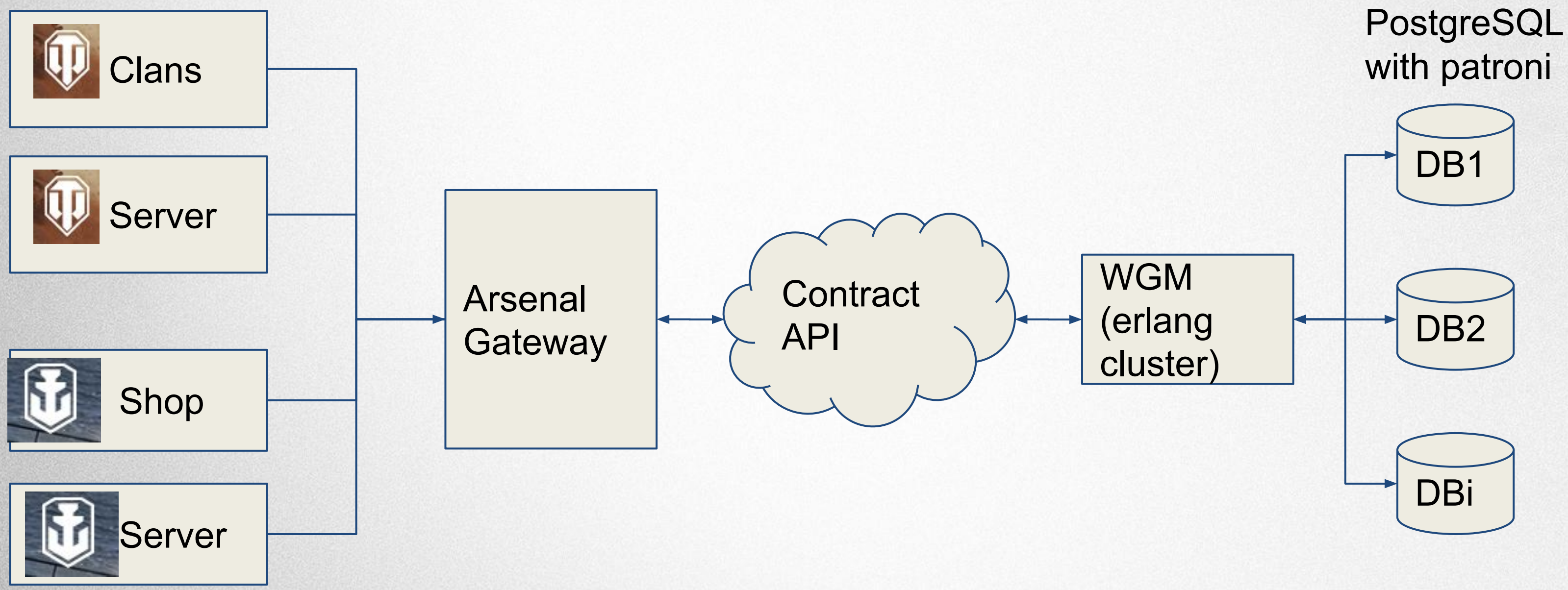
Agenda

- **OrioleDB intro from Mr. Korotkov**
- **Use-case - Wargaming Money(WGM)**
- **Functional tests and stabilization**
- **Performance tests**
- **Conclusion**

Arsenal platform

1) Connects games and players inside WG
2) Auth & accounts, **commerce**, distribution, **inventory**, customer support and a lot more capabilities
3) Hundreds of services for games, players and game publishers
4) 1000+ PostgreSQL databases, with HA on patroni for critical services
5) 200+ TB data in PostgreSQL

WGM: currency storage for games



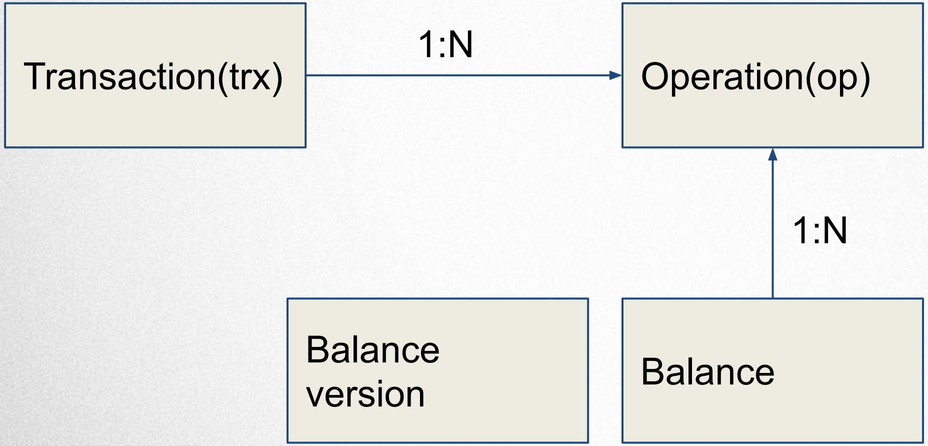
WGM: main traits

1) Low-latency and high performance
2) Ability to share currency balance between games
3) Track all paid currency deposits for revenue recognition
4) DB storage scaled horizontally using sharding

WGM: main use cases

1) Read currency balance
2) Change currency balance (+/-)
3) Change currency balance (+/-), 2 phase commit (begin + commit/rollback)
4) Exchange currency pairs with tracking paid/unpaid
5) Read currency balance in the past + finlog

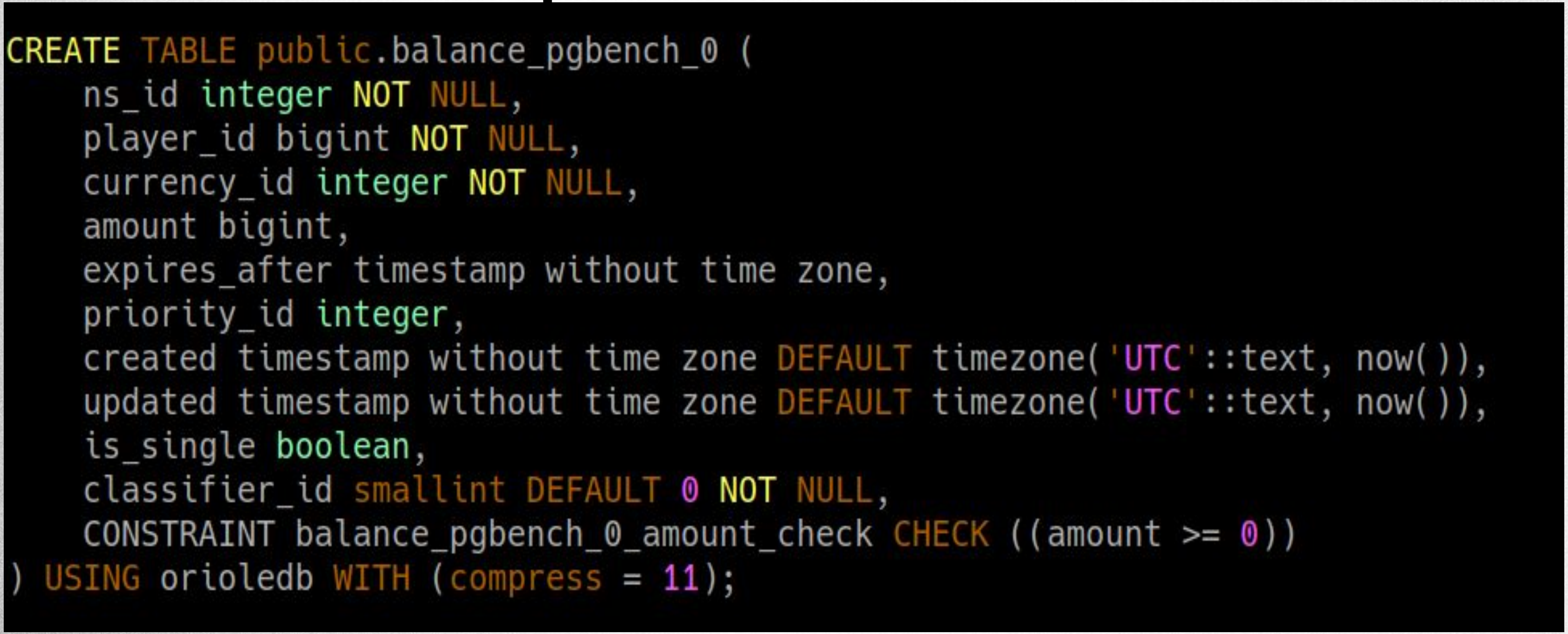
WGM: db schema



WGM: sharding schema

1) Each sharded table has numeric suffix from 0 to **shard_num**, e.g. trx_0, trx_1.
2) Shard_num is set on initial deploy and frozen
3) Tables are distributed on several physical databases according to config
4) Sharding algo - jump consistent hash

WGM: example of table



```
CREATE TABLE public.balance_pgbench_0 (
    ns_id integer NOT NULL,
    player_id bigint NOT NULL,
    currency_id integer NOT NULL,
    amount bigint,
    expires_after timestamp without time zone,
    priority_id integer,
    created timestamp without time zone DEFAULT timezone('UTC'::text, now()),
    updated timestamp without time zone DEFAULT timezone('UTC'::text, now()),
    is_single boolean,
    classifier_id smallint DEFAULT 0 NOT NULL,
    CONSTRAINT balance_pgbench_0_amount_check CHECK ((amount >= 0))
) USING orioledb WITH (compress = 11);
```

https://github.com/drednout/pgcon2022_orioledb_wgm_bench

WGM: why sharding on app?

Pros	Cons
1) Provide horizontal scaling 2) Avoid some PostgreSQL limitations: e.g. wraparound, high-contention on sequences 3) Easier maintenance(pg_repack,partial dump, etc.) 4) Improve performance in many cases	1) Need extra logic on app level 2) Need custom db migration tool

WGM: migration tool sdbmigrate

1) **sdbmigrate** supports sharding schema used by WGM out of the box
2) shard ranges may be set manually or calculated by **sdbmigrate**
3) migrations may be written in plain SQL or Python code
4) written in Python, 2.7 and 3.8+ python are supported
5) many small tasty features: schema versioning, dry run transactional migrations, variables substitution, etc.
6) PostgreSQL and MySQL databases are supported

https://github.com/wgnet/sdbmigrate

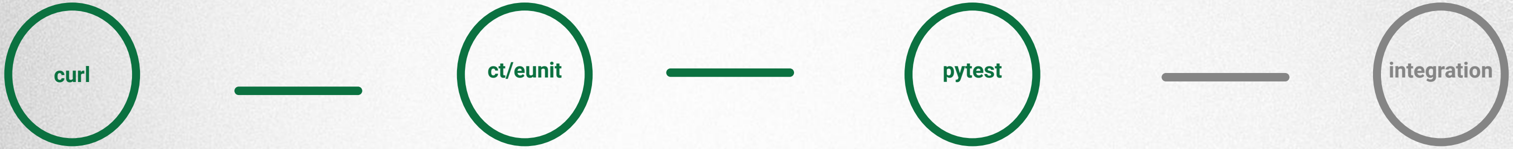
WGM+orioledb: what for?

1) Compression: tens of servers, terabytes of data
2) Heavy-write workload needs I/O decreasing, especially on big data sets
3) A lot of design fixes in PostgreSQL looks promising
4) It’s funny to develop/test databases :)

WGM+orioledb: test plan

1) Smoke tests in Vagrant env + fixes
2) Isolation functional tests + fixes
3) Performance tests via pgbench scripts + optimizations
4) Performance tests via app + optimizations
5) Long performance tests + optimizations
6) Integration functional tests
7) Connect orioledb as logical replica to real PostgreSQL database

Functional tests results



curl

Smoke tests

run locally set of small tests

ct/eunit

Erlang eunit and common tests

pytest

Set of NPQA tests For all major features

integration

Set of external test suites + manual QA tests

Problem fixed in orioledb:

1) Problem with updating jsonb
2) Problem with selects on partial indices
3) And a lot more, including: https://github.com/orioledb/orioledb/issues?q=is%3Aissue+author%3Adrednout+is%3Aclosed+

Performance tests via pgbench

1) Prepare simplified DB schema for testing changing/reading balance (80/20)
2) Create pgbench scripts for simulating prod-like transactions - write/read
3) Run tests for 1 hour in different profiles and measure throughput, latency, CPU and disk stats.

All scripts and configs are here:
https://github.com/drednout/pgcon2022_orioledb_wgm_bench

Prepare DB: test profiles

1) Orioledb no compression - all tables with orioledb storage, no compression.
2) Orioledb medium compression - all tables with orioledb storage, medium compression
3) Orioledb max compression - all tables with orioledb storage, medium compression
4) Orioledb hybrid with heap - balance and balance version tables with heap storage, transaction and operation with orioledb.
5) PostgreSQL heap - all tables with good old heap storage

Perftest stand: dedicated server

Kernel: 3.10.0-1160.11.1.el7.x86_64
OS: CentOS Linux release 7.9.2009
CPU: Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz, 56 cores
RAM: 384 GB
Disk: 2 TB SSD RAID 1
PostgreSQL: 13.6

Orioledb settings

```
default_table_access_method = 'orioledb'
orioledb.main_buffers = 12GB
orioledb.undo_buffers = 6GB
orioledb.checkpoint_completion_ratio = 1.0
orioledb.free_tree_buffers = 6GB
```

More info here:
https://github.com/orioledb/orioledb/blob/main/doc/usage.md

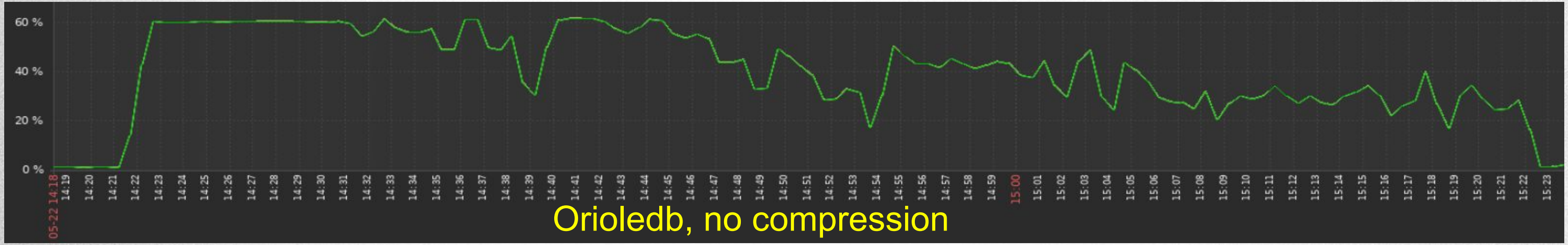
Write test methodology

1) Drop datadir, create empty DB
2) Run pgbench on 1 hour with custom script, simulating 1-phase transactions
3) During the test cpu/disk load is measured periodically via zabbix
4) Stop PostgreSQL and measure data size on disk

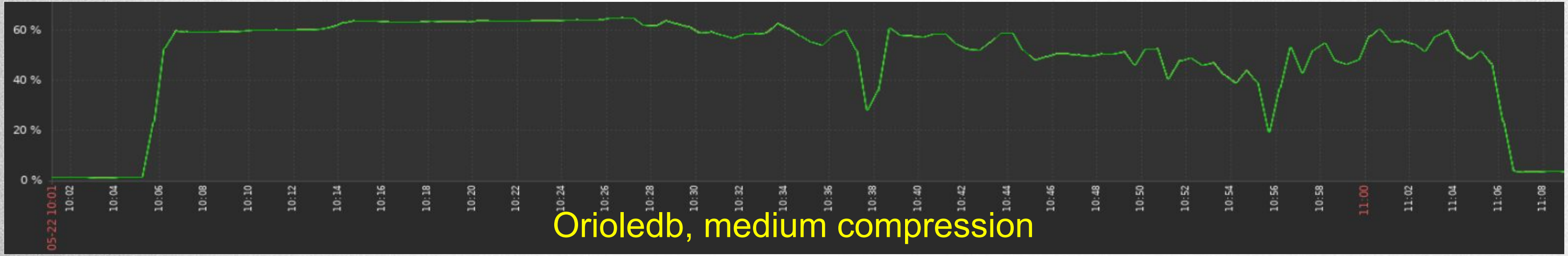
Write test results

Test	Throughput, tps	Latency avg, ms	Latency stddev, ms	Data on disk, GB	Undo log, GB
Orioledb, no compression	28141	1.75	10.16	79	7.3
Orioledb, medium compression	28580	1.73	6.62	26	15
Orioledb, max compression	24784	2.00	3.24	17	80
Orioledb, hybrid	30453	1.62	9.17	27.8	11
Heap(standard access method)	32502	1.511	6.60	53	-

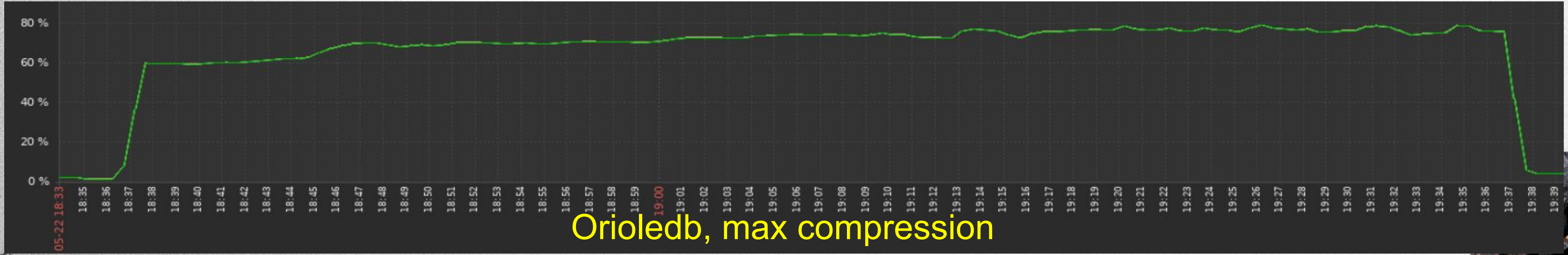
Write test results: cpu



Orioledb, no compression

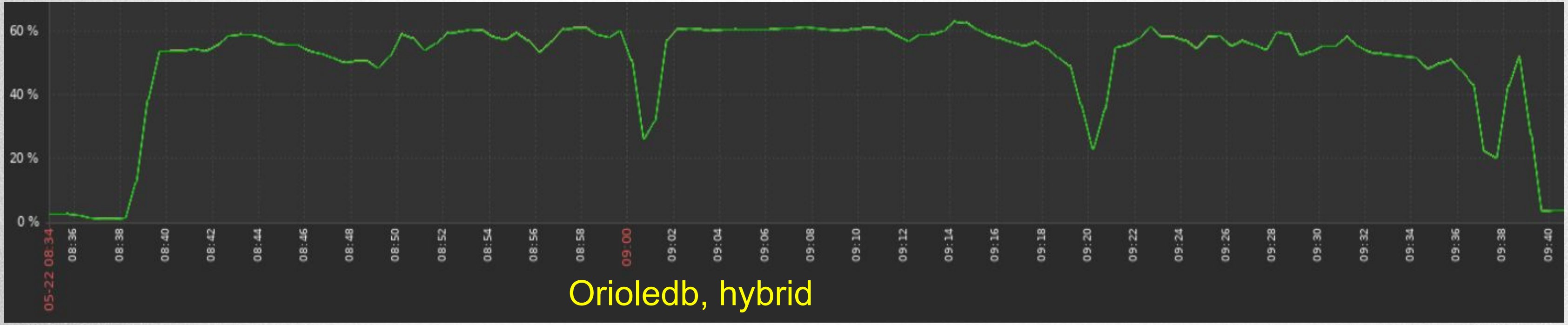


Orioledb, medium compression

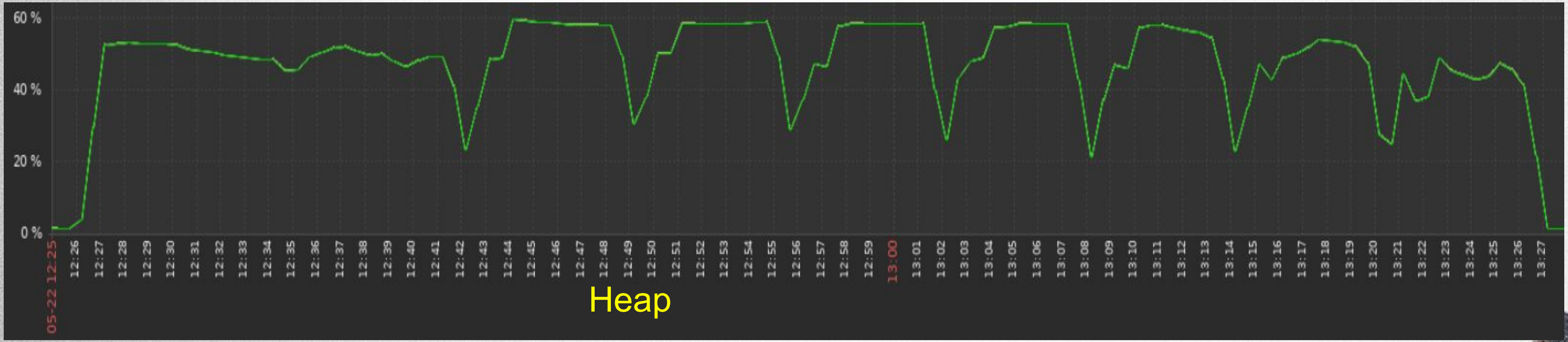


Orioledb, max compression

Write test results: cpu



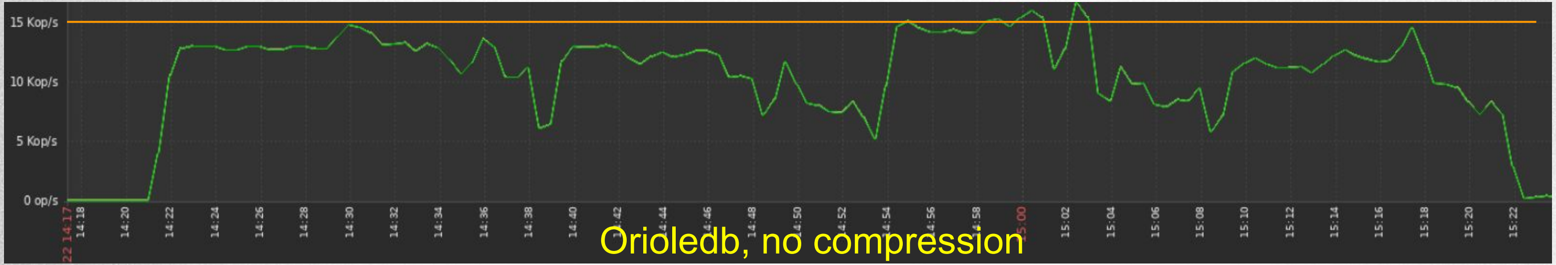
Orioledb, hybrid



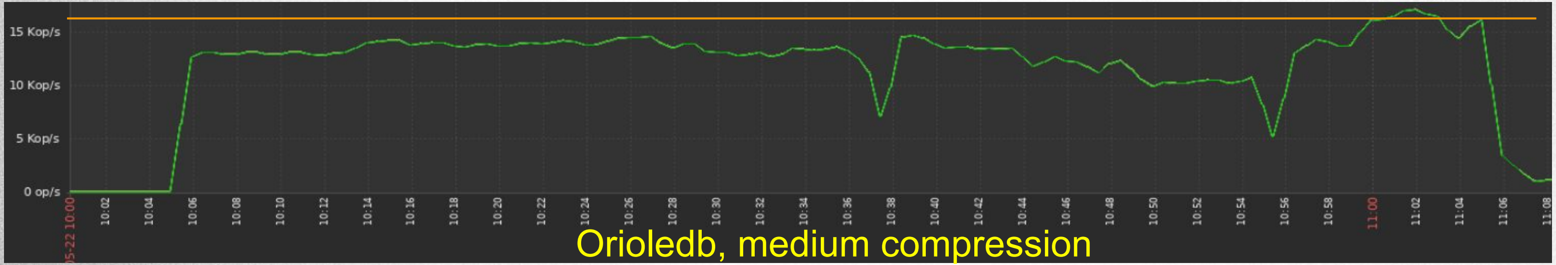
Heap

WARGAMING.NET
LET'S BATTLE

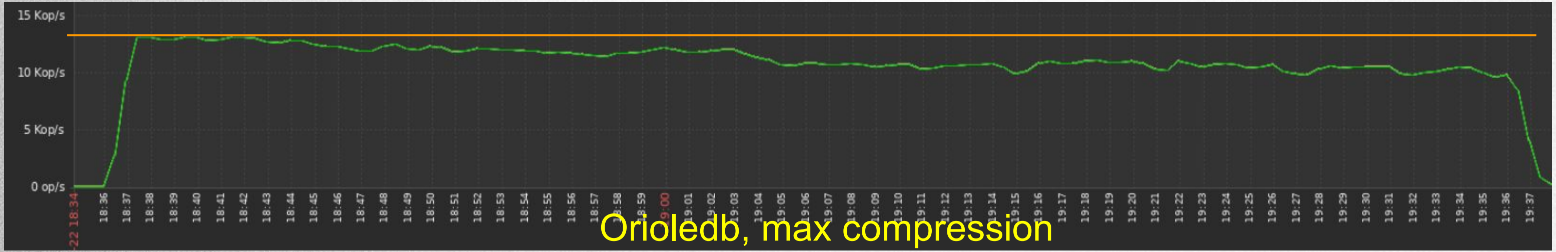
Write test results: disk iops



Orioledb, no compression

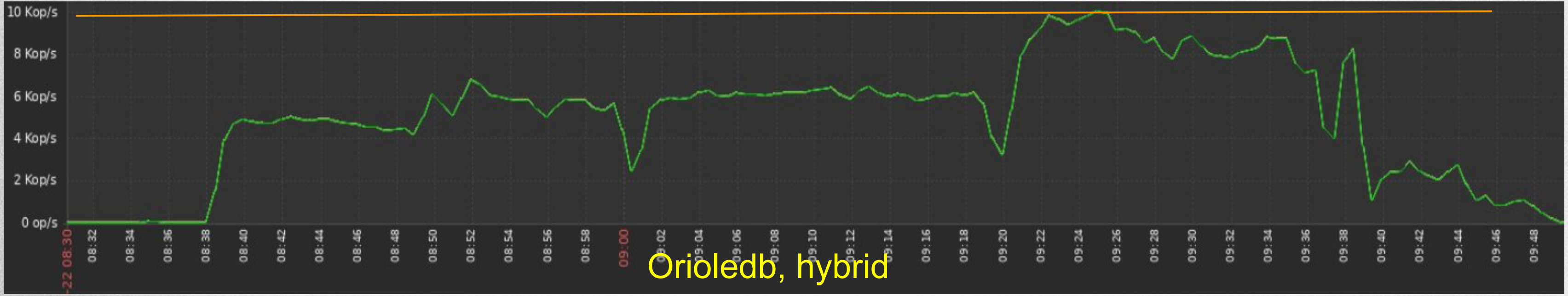


Orioledb, medium compression

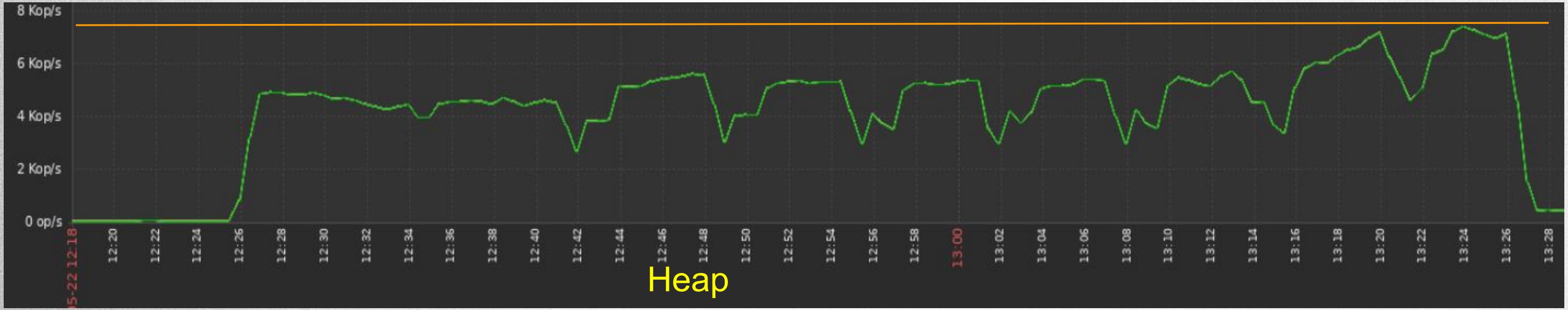


Orioledb, max compression

Write test results: disk iops



Orioledb, hybrid



Heap

Write test: short conclusion

1) Heap is the winner, orioledb in hybrid mode with second place has 7% less TPS
2) Orioledb in compressed mode consumes 50-66% disk less than heap
3) Data in hybrid mode consumes 48% less disk space than heap, almost like full compression
4) Orioledb produce extra disk load(IOPS) in comparison with heap
5) Orioledb with max compression level behaves pretty strange - low perf, big undo log, a lot of IOPS. Need extra investigation from dev team.

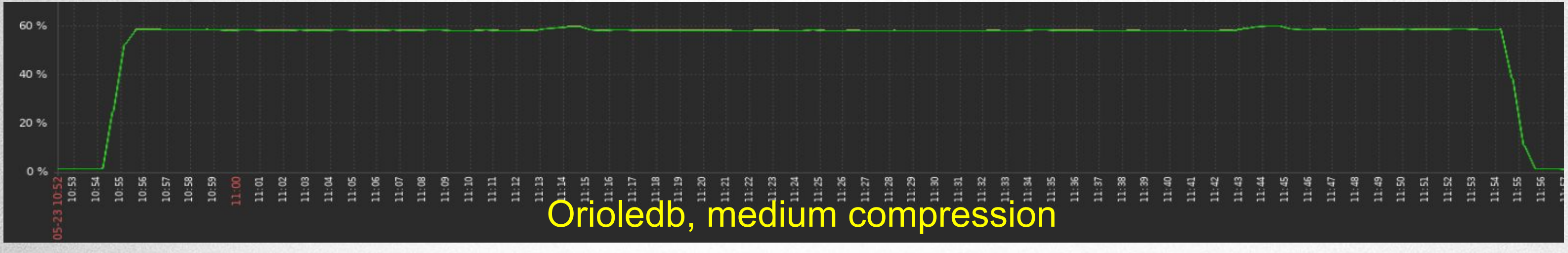
Read test methodology

1) Drop datadir, create empty DB
2) Run write test for 1 hour with 1-phase transactions, wait for finish
3) Run read test for 1 hour
4) During the test cpu/disk load is measured periodically via zabbix

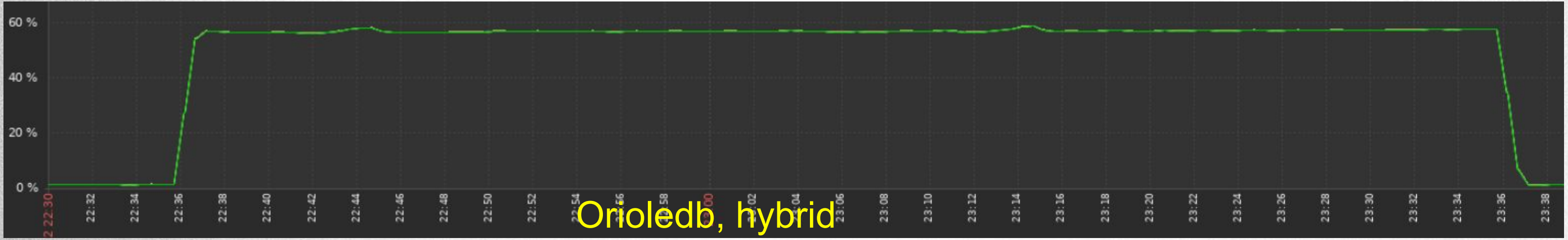
Read test results

Test	Throughput, tps	Latency, ms	Latency stddev, ms
Orioledb, medium compression	68367	0.70	0.44
Orioledb, hybrid	70805	0.68	0.15
Heap	69484	0.69	0.23

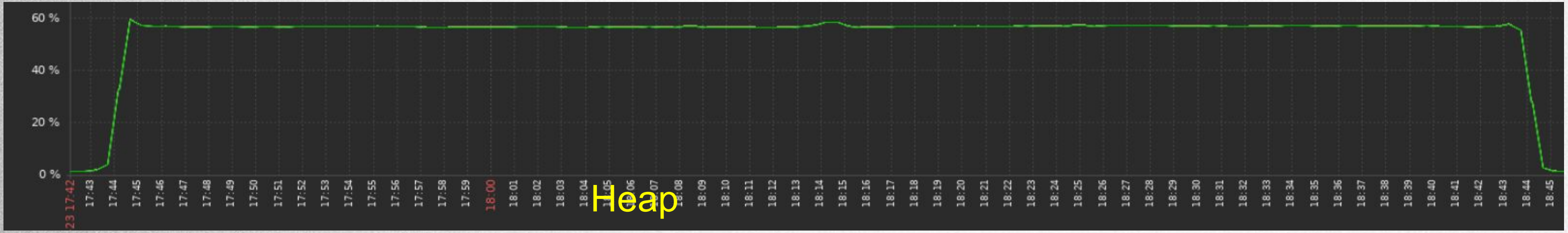
Read test results: cpu



Orioledb, medium compression

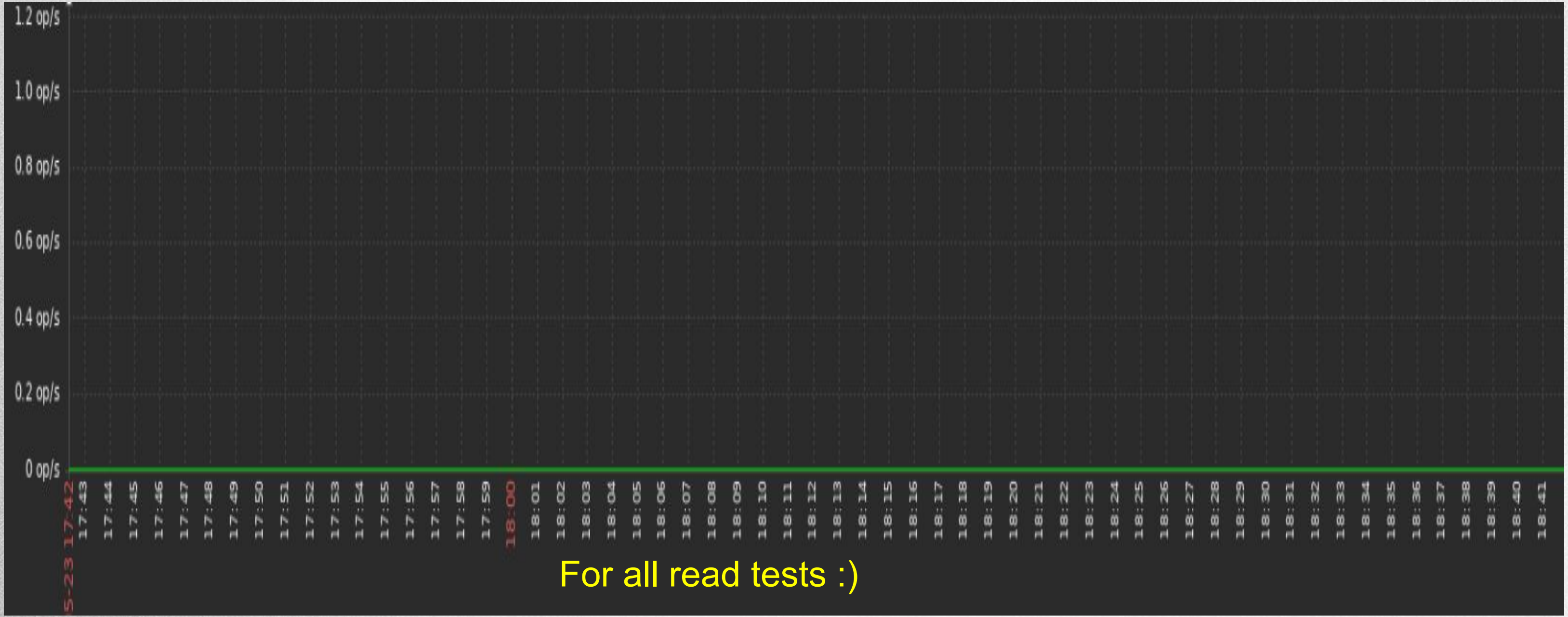


Orioledb, hybrid



Heap

Read test results: disk read iops



For all read tests :)

Read test: conclusion

1) good results both for heap and orioledb
2) Orioledb+heap is the winner(+2% in comparison with heap)
3) No read disk IOPS in this test - all data were in the RAM, it’s OK

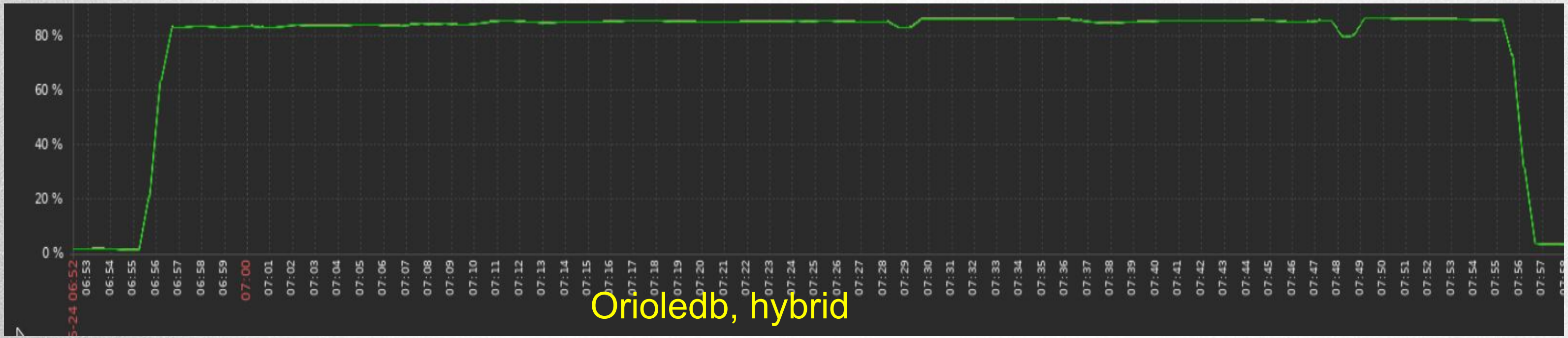
Mix read/write test

1) Drop datadir, create empty DB
2) Run write test for 1 hour with 1-phase transactions
3) Run read test for 1 hour simultaneously
4) During the test cpu/disk load is measured periodically via zabbix

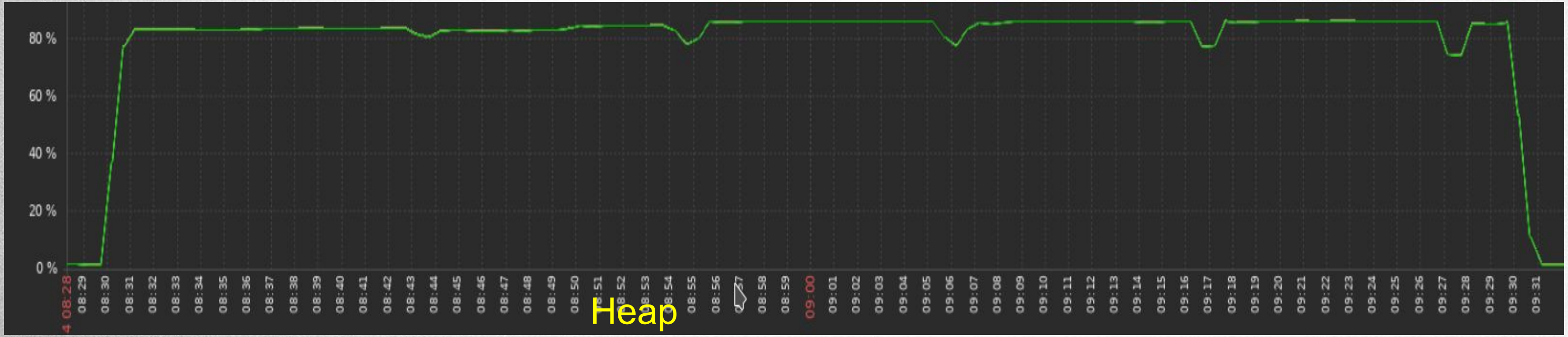
Mix test results

Test	Throughput, tps	Latency, ms	Latency stddev, ms
Orioledb, hybrid, write	20592	2.39	7.33
Orioledb, hybrid, read	49736	0.98	0.76
Heap, write	20912	2.35	3.9
Heap, read	51283	0.95	0.37

Mix test results: cpu

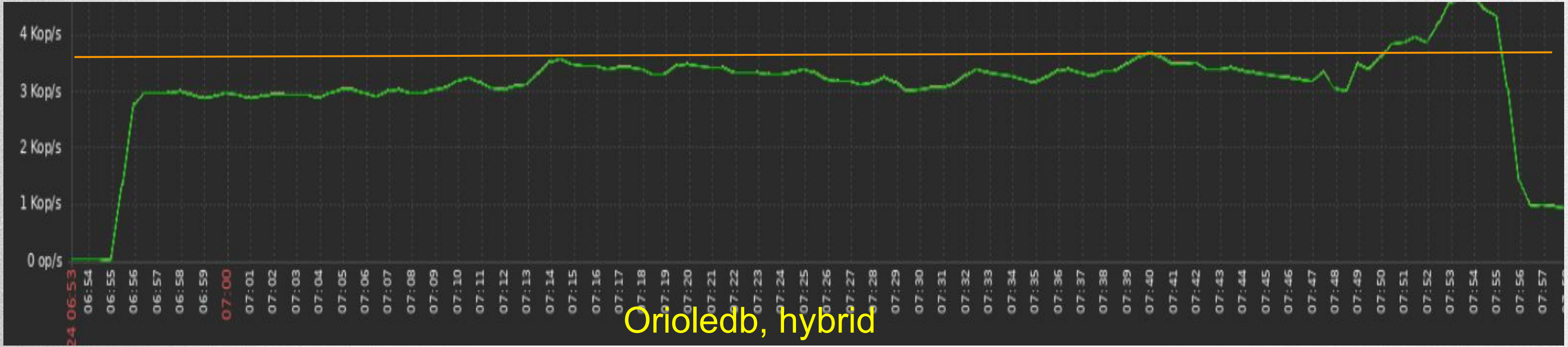


Orioledb, hybrid

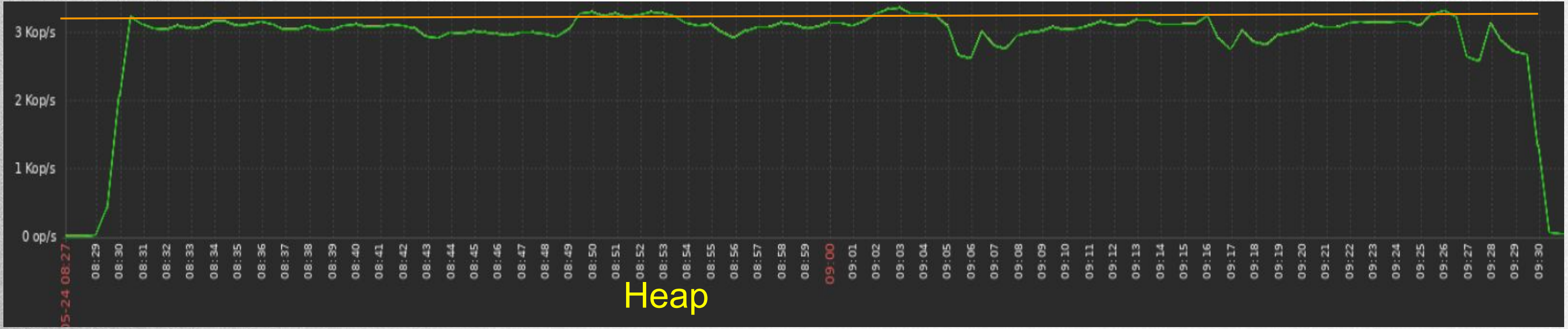


Heap

Mix test results: disk iops



Orioledb, hybrid



Heap

Mix test: conclusion

1) Simultaneous write and read tests shows very similar performance, number looks good
2) The number of IOPS is higher for orioledb in comparison with heap

Conclusion

1) Orioledb shows pretty good results on write/read/mix tests
2) Orioledb saves 48-66% of disk space in comparison with heap
3) The number of IOPS on heavy write load should be improved in orioledb
4) Some scenarios discover problems e.g. write test on max compression, need to investigate them and fix issues
5) Orioledb may be effectively used in hybrid mode - only for some tables
6) We are planning to continue testing orioledb+wgm
7) Orioledb shows good development pace and may be considered as production database in 2023-2024

Future steps

1) Improve performance in orioledb :)
2) Do performance tests with application
3) Do long performance tests to understand orioledb behavior on the long distance
4) Connect orioledb as a replica to production server