zalando

NOSQL FOR POSTGRESQL

BEST PRACTICES

DMITRY DOLGOV

09-27-2017

zalando

zalando

➔ Jsonb internals and performance-related factors

- ➔ Jsonb internals and performance-related factors
- ➔ Tricky queries

- ➔ Jsonb internals and performance-related factors
- ➔ Tricky queries
- ➔ Benchmarks

- ➔ Jsonb internals and performance-related factors
- ➔ Tricky queries
- ➔ Benchmarks
- ➔ How to shoot yourself in the foot

Live Long, and Prosper

- Han Solo

AWS EC2

m4.large instance

separate instance (database and generator)

16GB memory, 4 core 2.3GHz

Ubuntu 16.04

Same VPC and placement group

AMI that supports HVM virtualization type

at least 4 rounds of benchmark

PostgreSQL 9.6.3/10

MySQL 5.7.9/8.0

MongoDB 3.4.4

YCSB 0.13

10^6 rows and operations

AWS EC2

Configuration

shared_buffers
effective_cache_size
max_wal_size
innodb_buffer_pool_size
innodb_log_file_size
write concern level (journaled or transaction_sync)
checkpoint
eviction

Document types

“simple” document
10 key/value pairs (100 characters)

“large” document
100 key/value pairs (200 characters)

“complex” document
100 keys, 3 nesting levels (100 characters)

Performance-related factors

Performance-related factors

➔ On-disk representation

Performance-related factors

- ➔ On-disk representation
- ➔ In-memory representation

Performance-related factors

- ➔ On-disk representation
- ➔ In-memory representation
- ➔ Indexing support

Indexing support

- Postgresql – single path, multiple paths, entire document
- MongoDB – single path, multiple paths
- MySQL – virtual columns, single path, multiple paths

PG indexing details

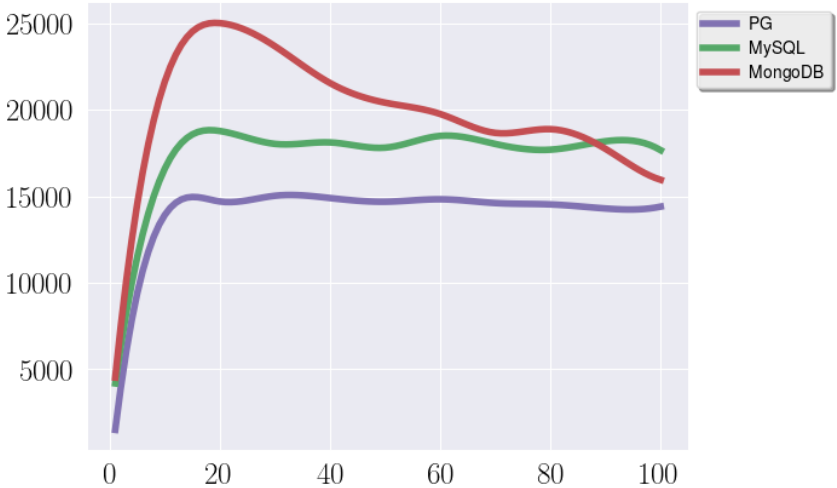
- ➔ jsonb_path
- ➔ jsonb_path_ops

Select, GIN

”simple” document

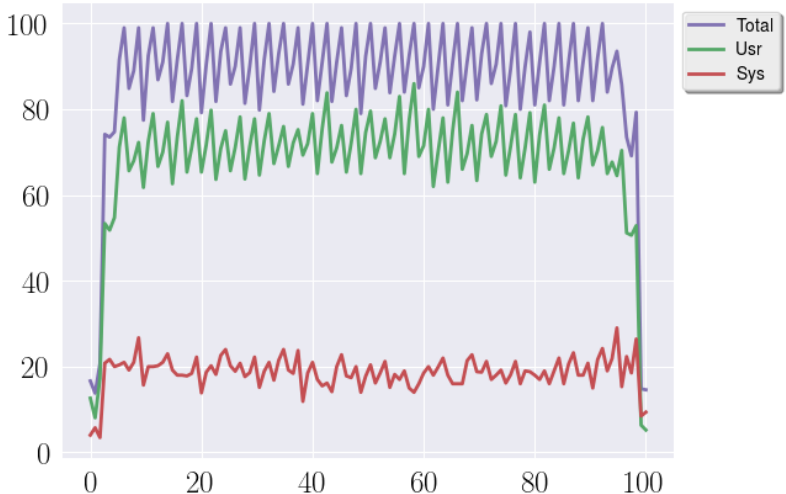
jsonb_path_ops

Throughput (ops/sec)



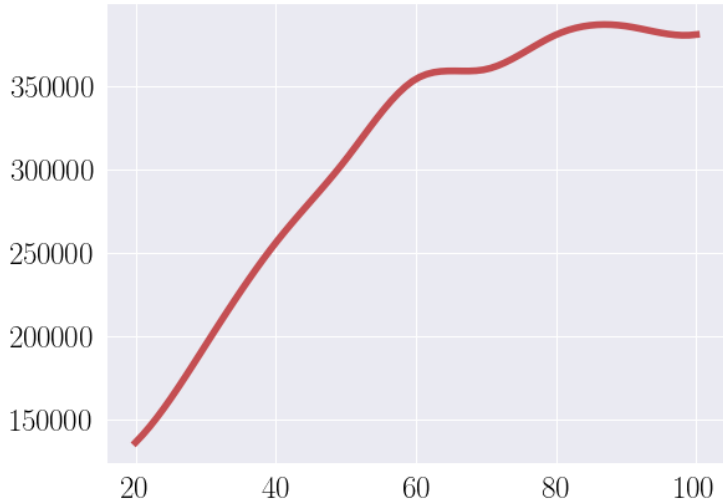
zalando

CPU%

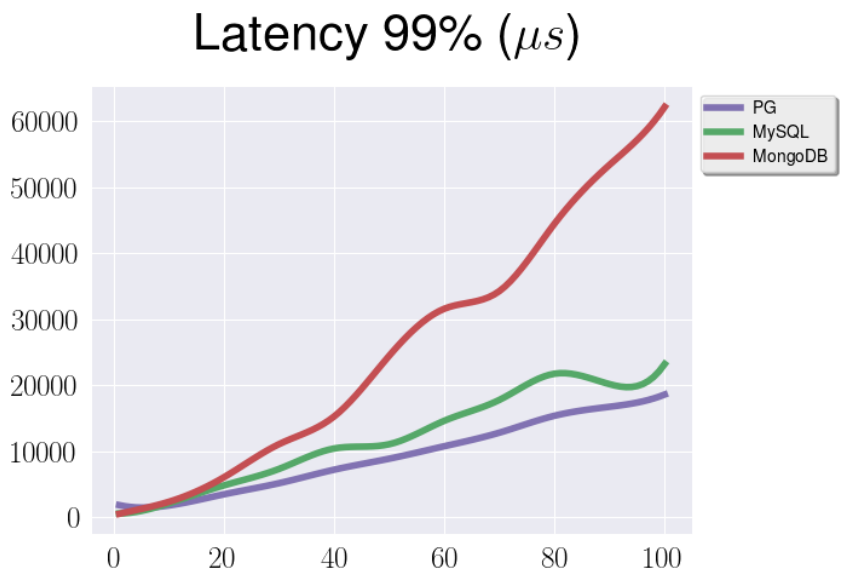


zalando

CPU migrations (MongoDB)



zalando


Latency 99% (μs)
60000
50000
40000
30000
20000
10000
0
0
20
40
60
80
100
PG
MySQL
MongoDB

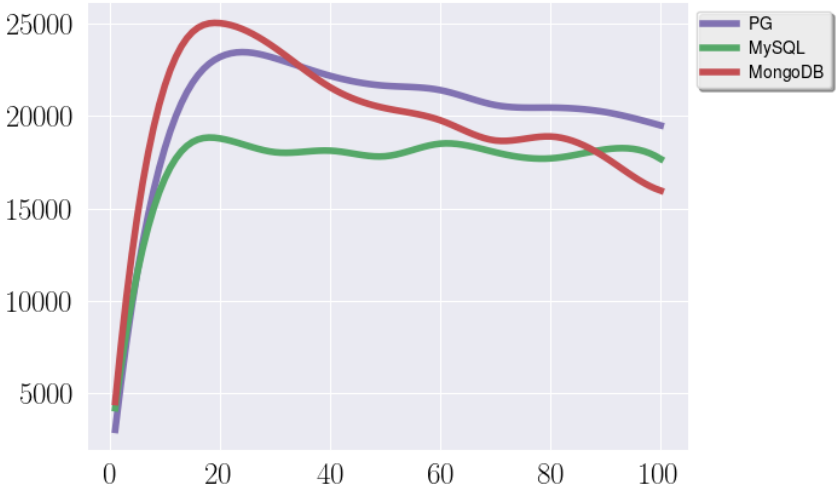
zalando

Select, BTree

”simple” document

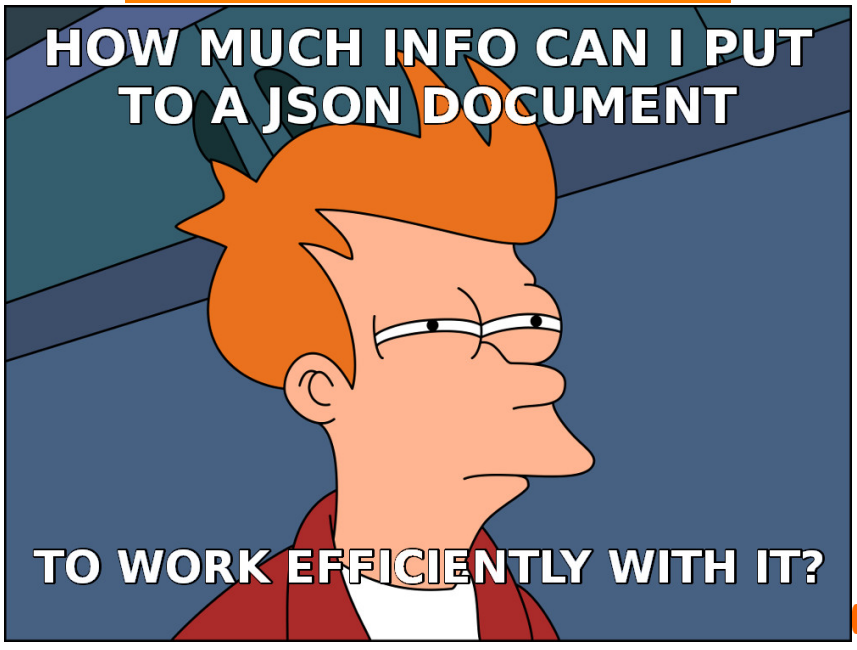
btree

Throughput (ops/sec)

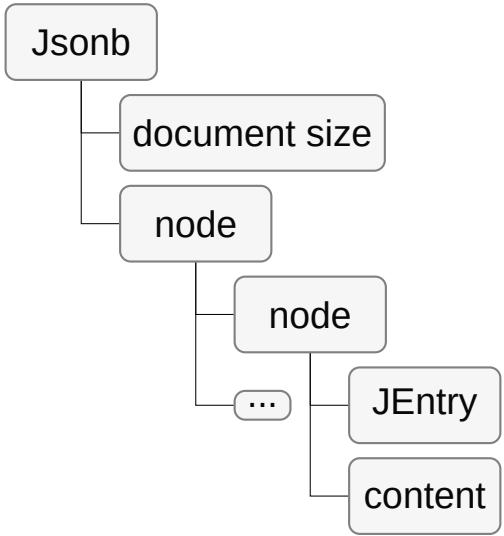


zalando

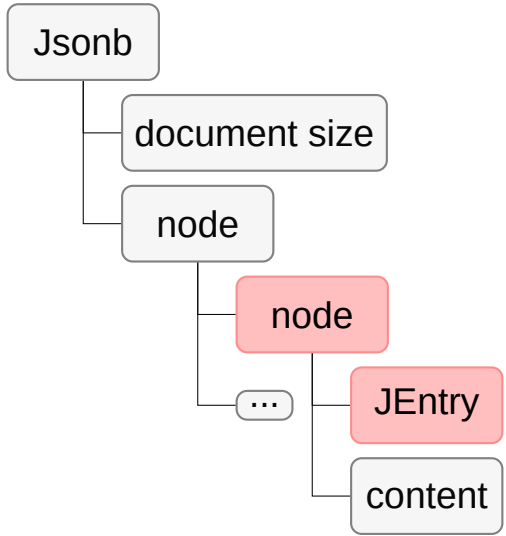
Internals


HOW MUCH INFO CAN I PUT TO A JSON DOCUMENT
TO WORK EFFICIENTLY WITH IT?

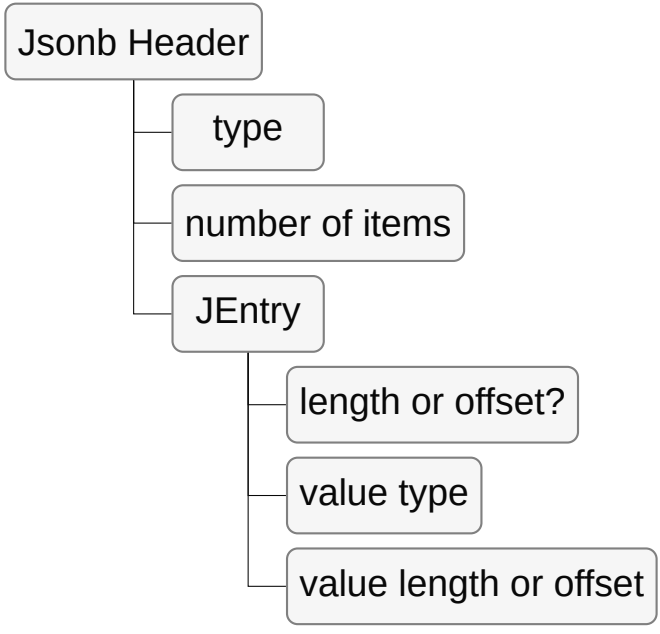
zalando



zalando



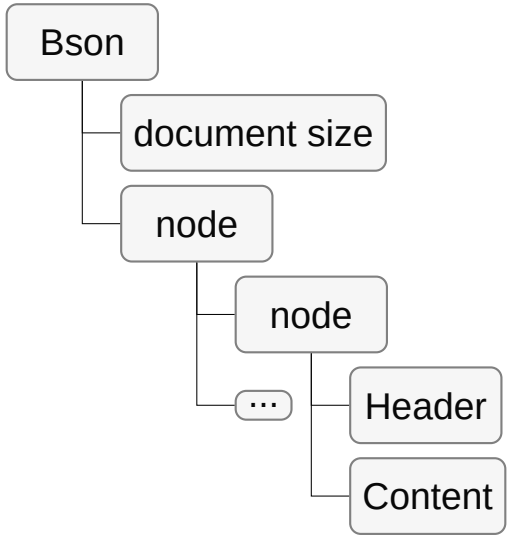
zalando


Jsonb Header
type
number of items
JEntry
length or offset?
value type
value length or offset

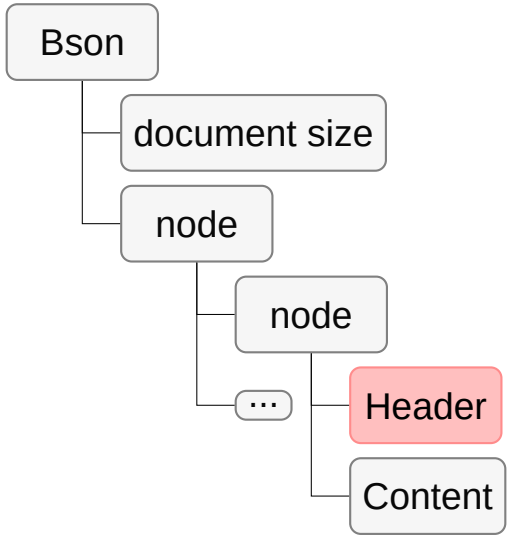
zalando

JB_OFFSET_STRIDE

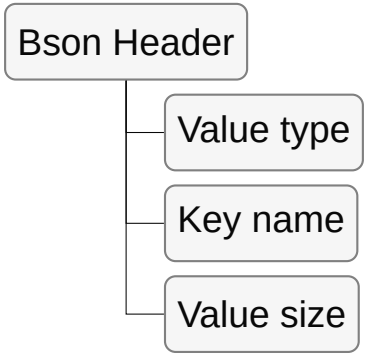
- JEntry may contains a value lenght or offset
- Offset = access speed
- Length = compressibility
- Every **JB_OFFSET_STRIDE**’th JEntry contains an offset
- Rest of them contain length



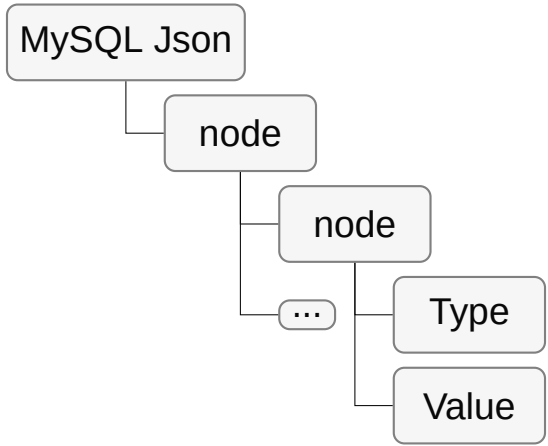
zalando



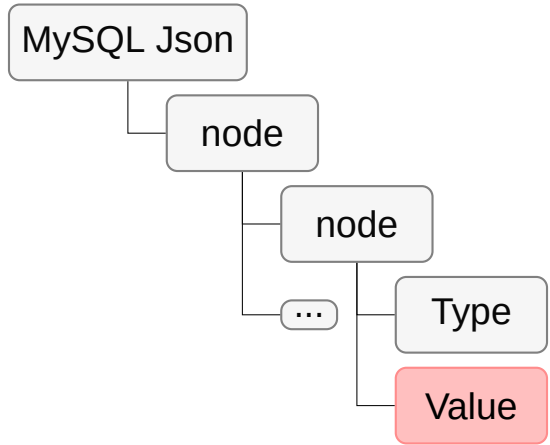
zalando



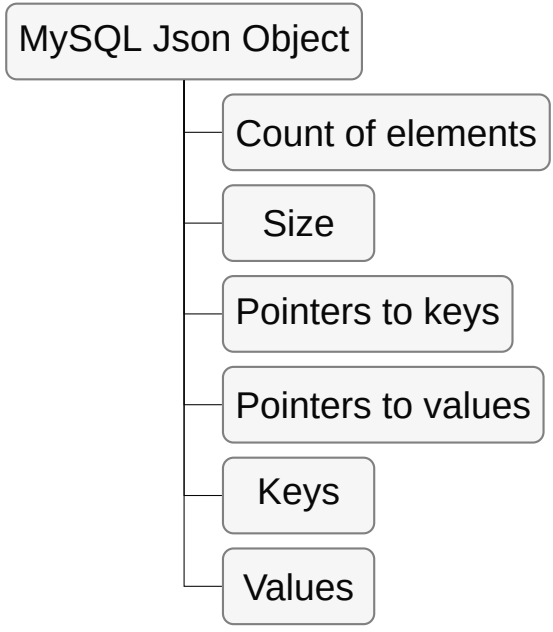
zalando



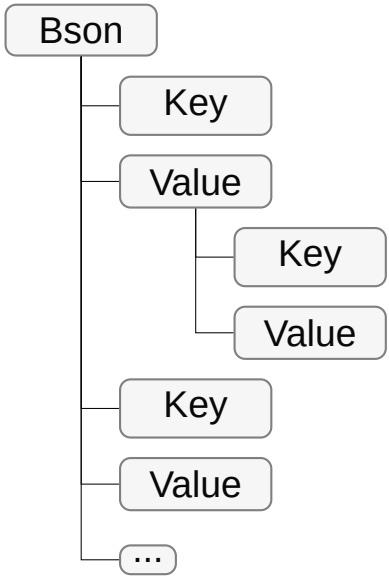
zalando

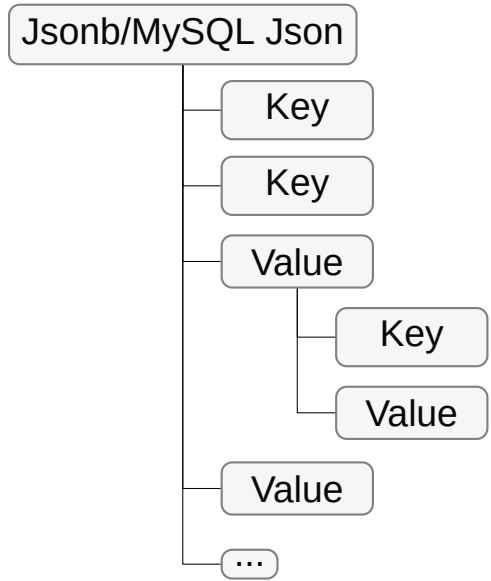


zalando


MySQL Json Object
Count of elements
Size
Pointers to keys
Pointers to values
Keys
Values

zalando


Bson
Key
Value
Key
Value
Key
Value
…


Jsonb/MySQL Json
Key
Key
Value
Key
Value
Value
…

zalando

```
{"a": 3, "b": "xyz"}
```

```
select pg_relation_filepath(oid),
relpages from pg_class
where relname = 'table_name';

 pg_relation_filepath | relpages
----------------------+----------
 base/40960/325477    |        0
(1 row)
```

```
\x10\x03\x00\x00\x00ab\x00\x00\x00\x00\x00\x00\x80\x03\x00xyz\x00\x00\x00\x00
```


zalando

```
bson.dumps({"a": 3, "b": u"xyz"})
```

```
\x17\x00\x00\x00\x10a\x00\x03\x00\x00\x00\x02b\x00\x04\x00\x00\x00xyz\x00\x00
```


zalando

```
$ hexdump -C database/table.ibd
```

```
\x00\x02\x00\x18\x00\x12\x00\x01\x00\x13\x00\x01\x00\x05\x03\x00\x0c\x14\x00ab\x03xyz\x00
```


zalando

Alignment

Variable-length portion is aligned to a 4-byte

```
insert into test
values('{"a": "aa", "b": 1}');
```

```
abaa\x20\x00\x00\x00\x00\x80\x01\x00
```

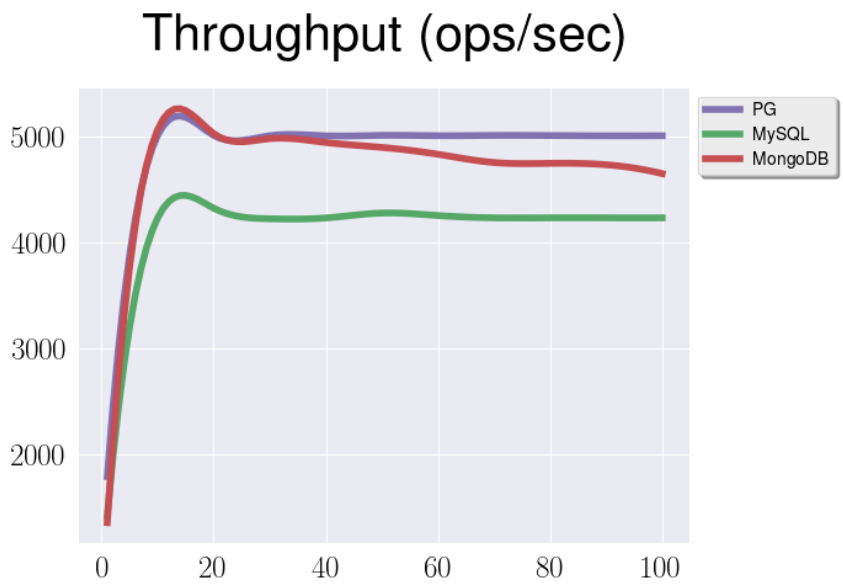
```
insert into test
values('{"a": 1, "b": "aa"}');
```

```
\x00\x00ab\x00\x00\x20\x00\x00\x00\x00\x80\x01\x00aa
```

Select, BTree

”complex” document

btree


Throughput (ops/sec)
5000
4000
3000
2000
0
20
40
60
80
100
PG
MySQL
MongoDB

zalando

TOAST_TUPLE_THRESHOLD

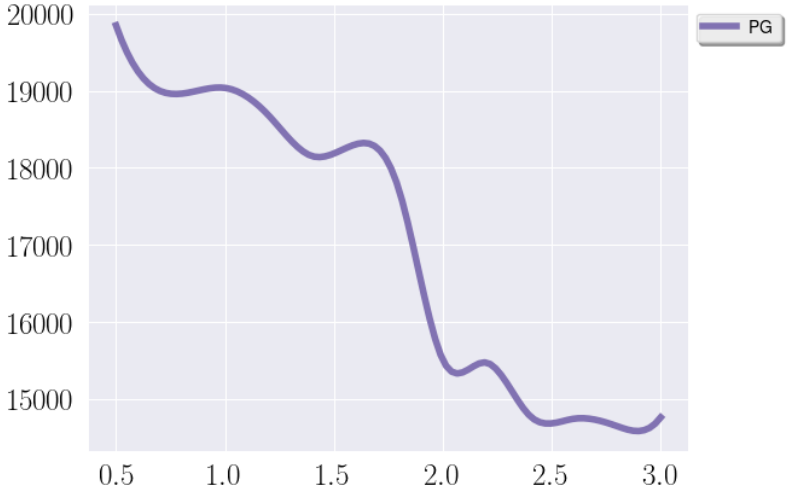
”simple” document

40 threads

different document size

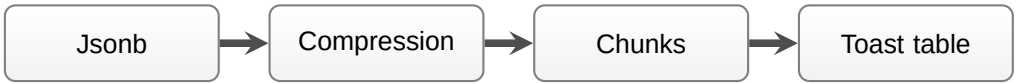
select

Throughput, 40 clients



zalando

TOAST



- ➔ TOAST_TUPLE_THRESHOLD bytes (normally 2 kB)
- ➔ PostgreSQL and MySQL use LZ variation
- ➔ MongoDB uses snappy block compression

In-memory representation

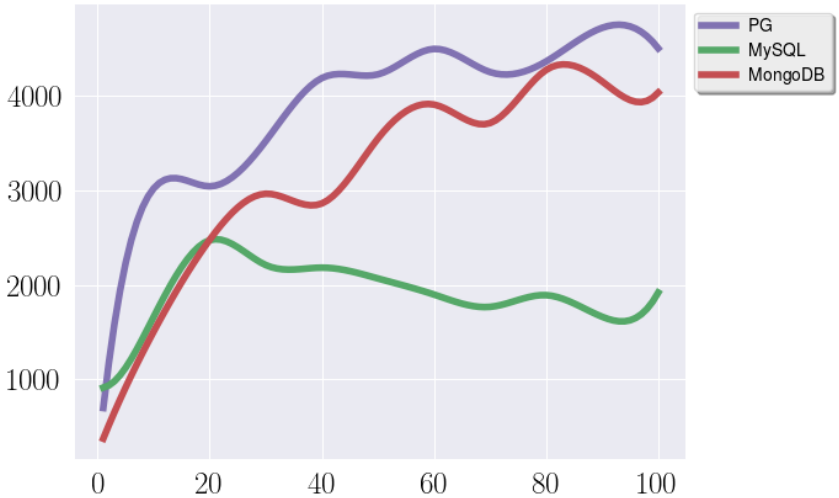
- ➔ Tree-like representation (JsonbValue, Document, Json_dom)
- ➔ Little bit more expensive but more convenient to work with
- ➔ Mostly in use to modify data (except MySQL)
- ➔ Most of the read operations use on-disk representation

Insert

”simple” document

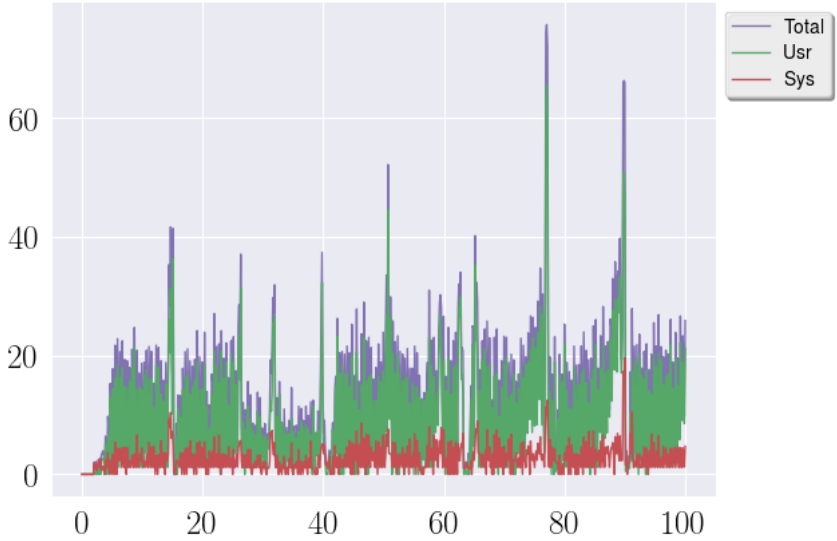
journaled

Throughput (ops/sec)



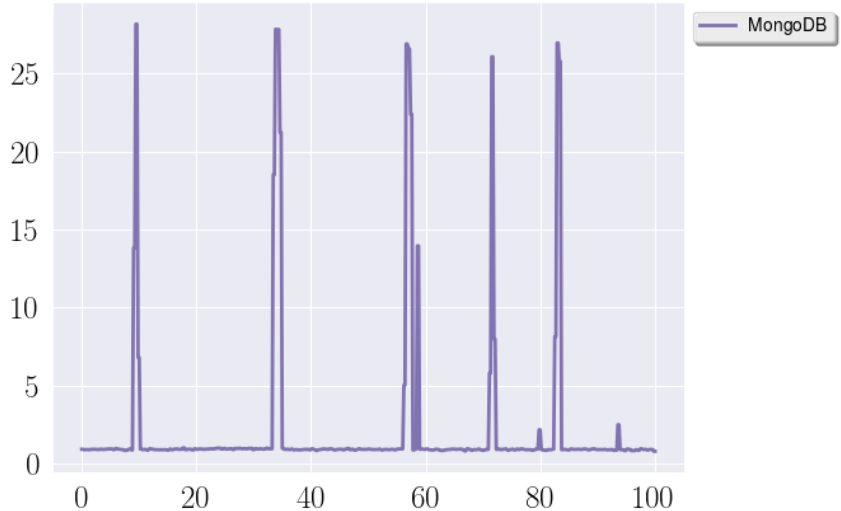
zalando

CPU%



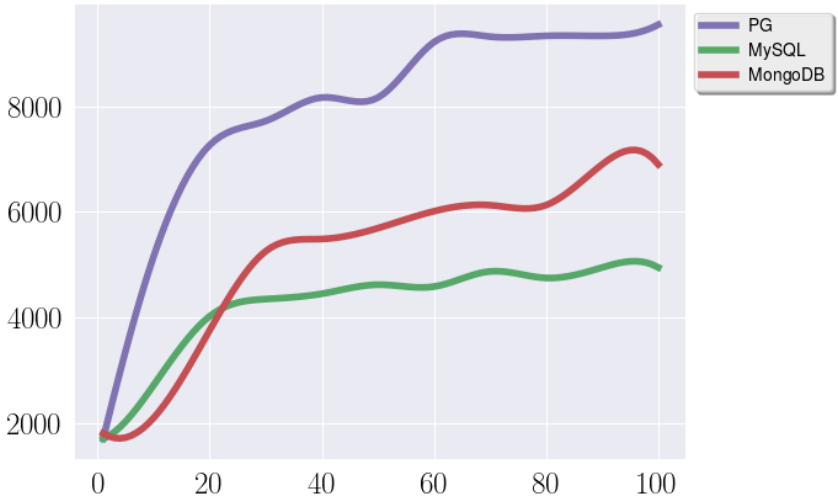
zalando

IO queue size



zalando

Throughput (ops/sec)



zalando

zalando

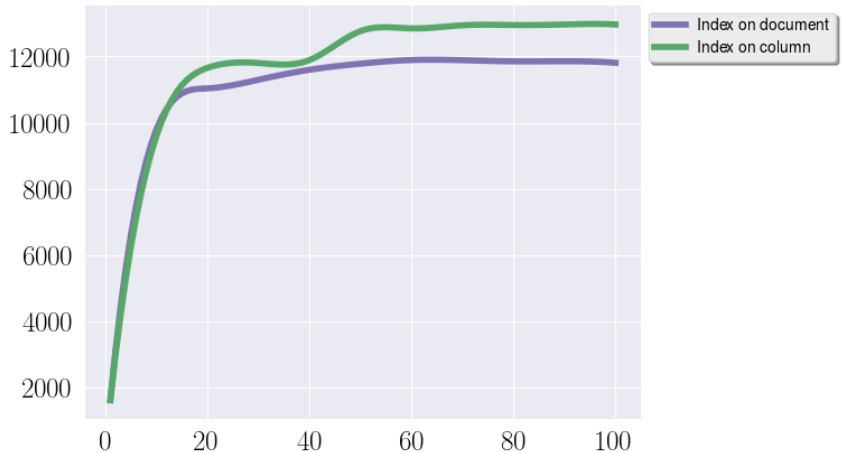
- ➔ Update one field of a document
- ➔ DETOAST of a document
  (select, constraints, procedures etc.)
- ➔ Reindex of an entire document

Index update

”simple” document

Update one field

PG, Throughput (ops/sec)



zalando

Update 50%, Select 50%

”simple” document

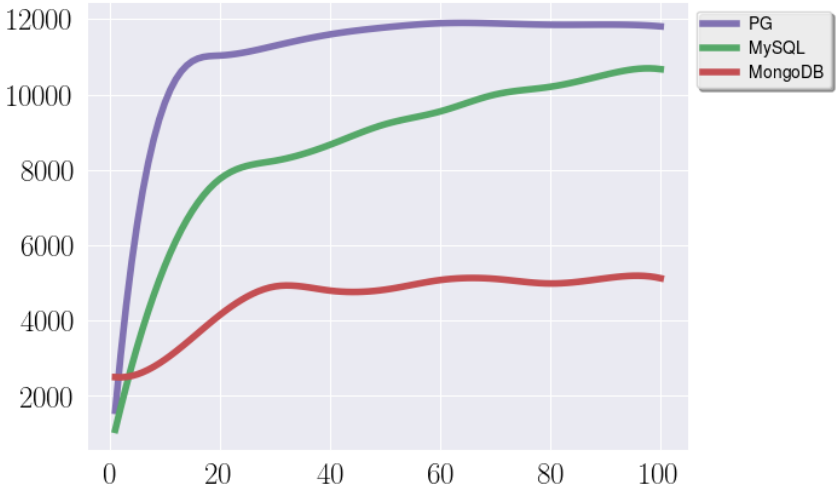
Update one field

journaled

max wal size 5GB

zalando

Throughput (ops/sec)



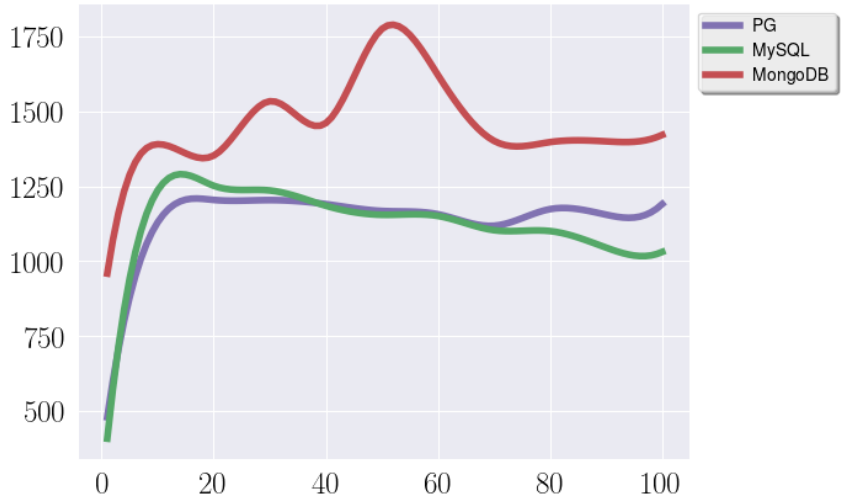
zalando

Update 50%, Select 50%

”large” document

Update one field

Throughput (ops/sec)



zalando

Document slice

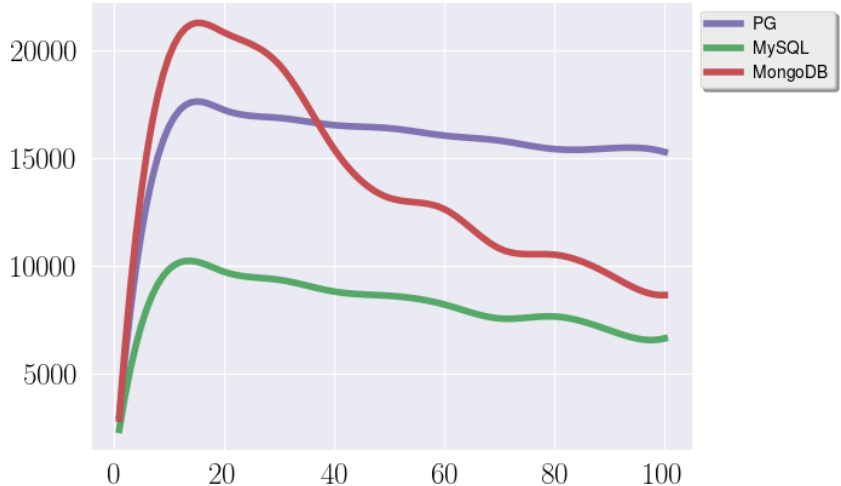
”large” document

One field from a document

```
select data→'key1'→'key2' from table;

select data→'key1', data→'key2' from table;
```

Throughput (ops/sec)



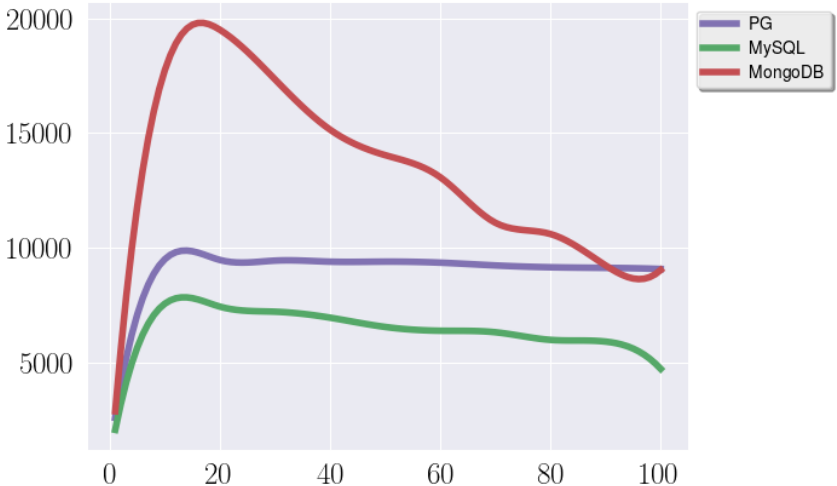
zalando

Document slice

”large” document

10 fields from a document

Throughput (ops/sec)



zalando

Solutions?

- ➔ set storage external
- ➔ different query

Queries

Pitfalls

- ➔ No Json path out of the box (jsquery, SQL/JSON)
- ➔ Queries with an array somewhere in the middle
- ➔ Iterating through document
- ➔ Update inside document

Document slice

```
create type test as ("a" text, "b" text);
insert into test_jsonb
values('{"a": 1, "b": 2, "c": 3}');
select q.* from test_jsonb,
jsonb_populate_record(NULL::test, data) as q;
 a | b
---+---
 1 | 2
(1 row)
```

```
[{
    "items": [
        {"id": 1, "value": "aaa"},
        {"id": 2, "value": "bbb"}
    ]
}, {
    "items": [
        {"id": 3, "value": "aaa"},
        {"id": 4, "value": "bbb"}
    ]
}]
```

```
WITH items AS (
    SELECT jsonb_array_elements(data→'items')
    AS item FROM test
)
SELECT * FROM items
WHERE item-»'value' = 'aaa';
```

```
item
————————
 {"id": 1, "value": "aaa"}
 {"id": 3, "value": "aaa"}
(2 rows)
```

```
{
    "items": {
        "item1": {"status": true},
        "item2": {"status": true},
        "item3": {"status": false}
    }
}
```

```
WITH items AS (
    SELECT jsonb_each(data→'items')
    AS item FROM test
)
SELECT (item).key FROM items
WHERE (item).value-»'status' = 'true';
```

```
key
——-
item1
item2
(2 rows)
```

zalando

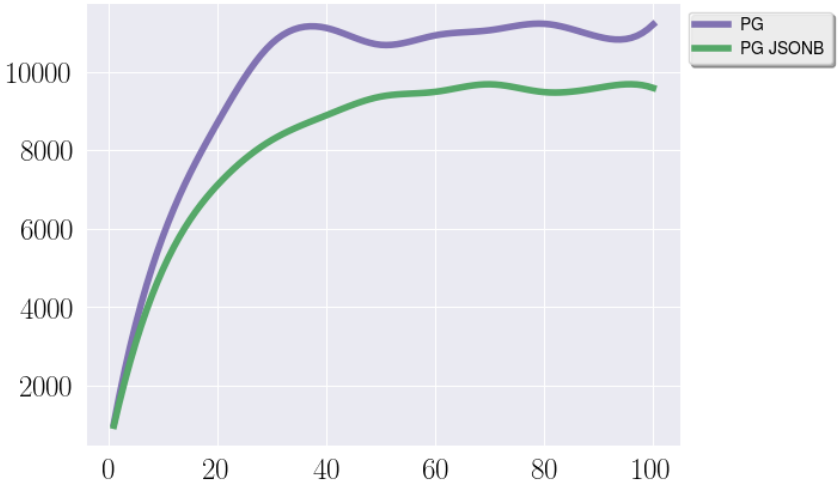
SQL vs JSONB

”simple” document

btree

insert

Throughput (ops/sec)



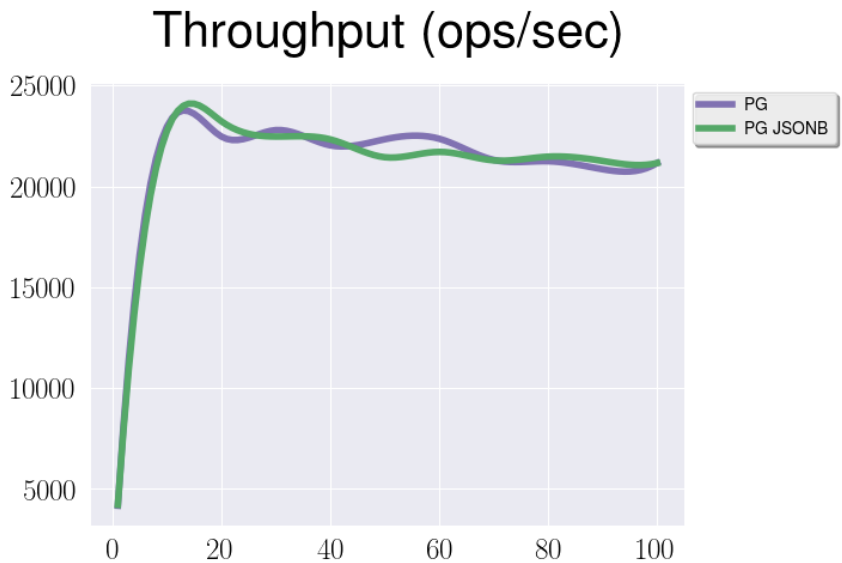
zalando

SQL vs JSONB

”simple” document

btree

select


Throughput (ops/sec)
25000
20000
15000
10000
5000
0
20
40
60
80
100
PG
PG JSONB

zalando

JSON vs JSONB

”simple” document

btree

insert


Throughput (ops/sec)
10000
8000
6000
4000
2000
0
20
40
60
80
100
PG JSON
PG JSONB

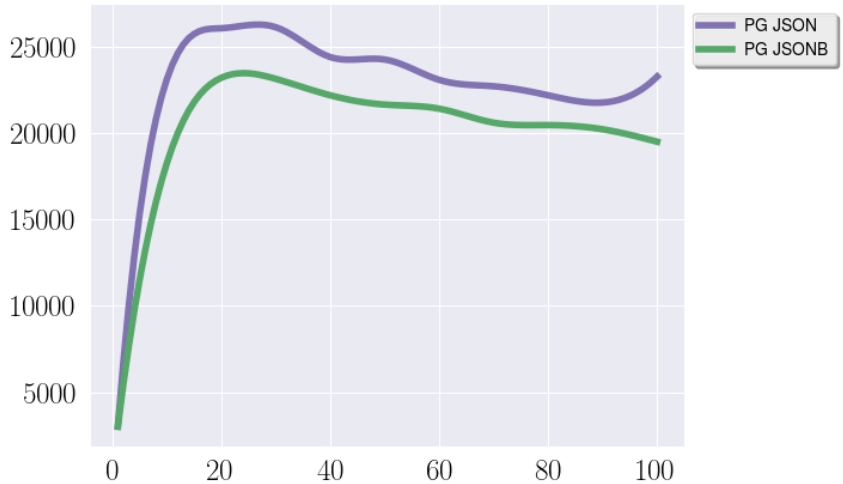
zalando

JSON vs JSONB

”simple” document

btree

select


Throughput (ops/sec)
25000
20000
15000
10000
5000
0
20
40
60
80
100
PG JSON
PG JSONB

zalando

➔ Jsonb is more that good for many use cases

- ➔ Jsonb is more that good for many use cases
- ➔ Reasons for performance difference is mostly database itself

- ➔ Jsonb is more that good for many use cases
- ➔ Reasons for performance difference is mostly database itself
- ➔ Benchmarks above are only ”hints”

- ➔ Jsonb is more that good for many use cases
- ➔ Reasons for performance difference is mostly database itself
- ➔ Benchmarks above are only ”hints”
- ➔ You need your own tests

Questions?

github.com/erthalion

@erthalion

9erthalion6 at gmail dot com

zalando