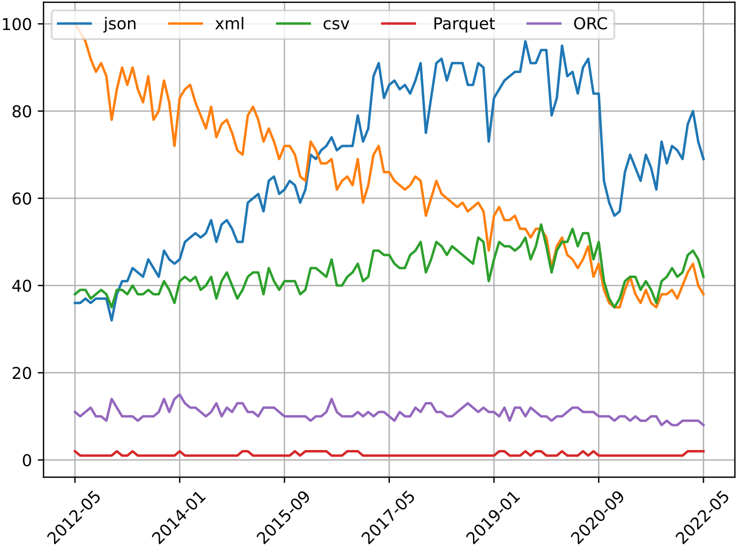
DeepBench – Benchmarking JSON Document Stores

Stefano Belloni, Daniel Ritter, Marco Schröder, Nils Rörup

SAP SE

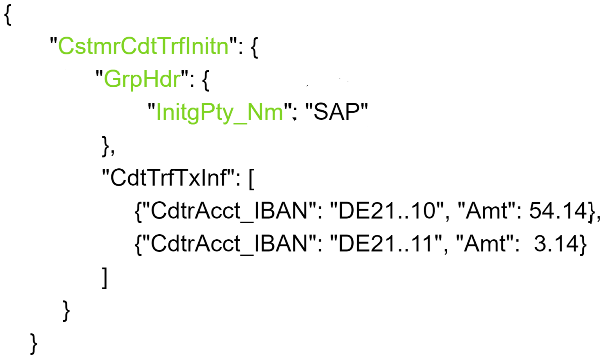
DBTest@Sigmod 2022

JavaScript Object Notation (short JSON)



(Google Trends, 5/22: Searches, worldwide; data exchange, storage format)

JSON and Query Example



```
{
    "CstmrCdtTrfInitn": {
        "GrpHdr": {
            "InitgPty_Nm": "SAP"
        },
        "CdtTrfTxInf": [
            {"CdtrAcct_IBAN": "DE21..10", "Amt": 54.14},
            {"CdtrAcct_IBAN": "DE21..11", "Amt":  3.14}
        ]
    }
}
```

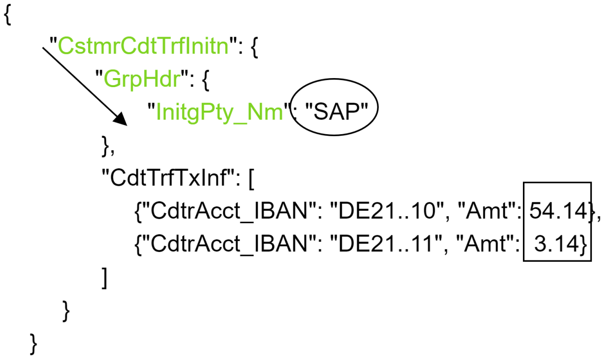
JSON and Query Example (Object)

```
SELECT  "CstmrCdtTrfInitn"
WHERE "CstmrCdtTrfInitn"."GrpHdr"."InitgPty_Nm" = 'SAP'
```

```
{
    "CstmrCdtTrfInitn": {
        "GrpHdr": {
            "InitgPty_Nm": "SAP"
        },
        "CdtTrfTxInf": [
            {"CdtrAcct_IBAN": "DE21..10", "Amt": 54.14},
            {"CdtrAcct_IBAN": "DE21..11", "Amt":  3.14}
        ]
    }
}
```

JSON and Query Example (Array UNNEST)

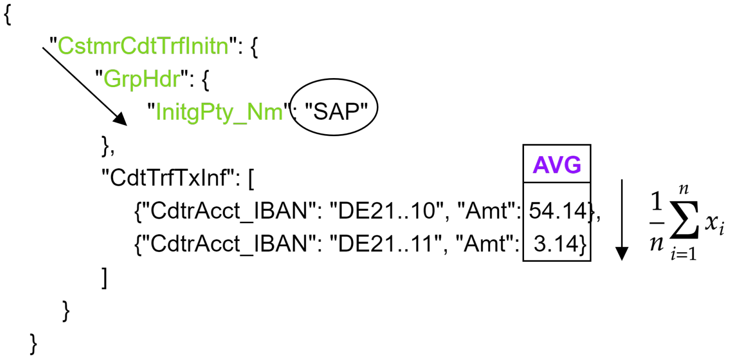
```
SELECT "unnested"."amt"
UNNEST "CstmrCdtTrfInitn"."CdtTrfTxInf" AS unnested
WHERE "CstmrCdtTrfInitn"."GrpHdr"."InitgPty_Nm" = 'SAP'
```



```
{
    "CstmrCdtTrfInitn": {
        "GrpHdr": {
            "InitgPty_Nm": "SAP"
        },
        "CdtTrfTxInf": [
            {"CdtrAcct_IBAN": "DE21..10", "Amt": 54.14},
            {"CdtrAcct_IBAN": "DE21..11", "Amt": 3.14}
        ]
    }
}
```

JSON and Query Example (Object, Array UNNEST, Aggregation)

```
SELECT AVG("unnested"."amt")
UNNEST "CstmrCdtTrfInitn"."CdtTrfTxInf" AS unnested
WHERE "CstmrCdtTrfInitn"."GrpHdr"."InitgPty_Nm" = 'SAP'
```



```
{
    "CstmrCdtTrfInitn": {
        "GrpHdr": {
            "InitgPty_Nm": "SAP"
        },
        "CdtTrfTxInf": [
            {"CdtrAcct_IBAN": "DE21..10", "Amt": 54.14},
            {"CdtrAcct_IBAN": "DE21..11", "Amt": 3.14}
        ]
    }
}
```

$$\frac{1}{n}\sum_{i=1}^{n} x_i$$

Benchmark Scale Dimensions

- concurrent users

Benchmark Scale Dimensions

- concurrent users
- result set size

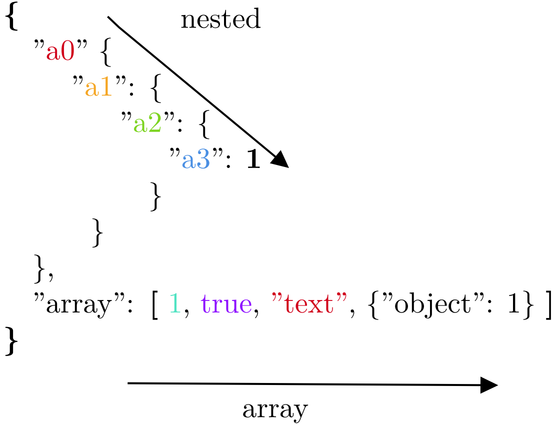
Benchmark Scale Dimensions

- concurrent users
- result set size
- query complexity

Benchmark Scale Dimensions

- concurrent users
- result set size
- query complexity
- JSON-specific dimensions
- ...

Benchmark Scale Dimensions

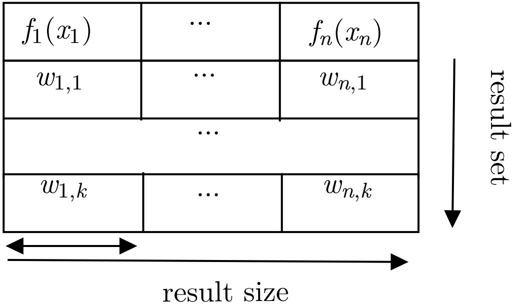


```
{
  ”a0” {
    ”a1”: {
      ”a2”: {
        ”a3”: 1
      }
    }
  },
  ”array”: [ 1, true, ”text”, {”object”: 1} ]
}
```

SELECT $f_1(x_1), \ldots \quad f_n(x_n)$

FROM collection

WHERE $P(x_1, \ldots x_2, y_1, \ldots, y_m)$



$f_1(x_1)$	$\cdots$	$f_n(x_n)$
$w_{1,1}$	$\cdots$	$w_{n,1}$
$\cdots$		
$w_{1,k}$	$\cdots$	$w_{n,k}$

- Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In SoCC. ACM, 143–154.
- Shiva Jahangiri. 2021. Wisconsin Benchmark Data Generator: To JSON and Beyond. In SIGMOD. ACM, 2887–2889

JSON Document Store Benchmarking Challenges

- Current benchmarking practices focus on YCSB (key-value benchmark, no nesting), and TPC-C (transactions)
- Recent advances in generating JSON data (Wisconsin benchmark)
- JSON-specific benchmark required for object and array data, and query / workload dimensions

- Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In SoCC. ACM, 143–154.
- Asya Kamsky. 2019. Adapting TPC-C Benchmark to Measure Performance of Multi-Document Transactions in MongoDB. Proc. VLDB Endow. 12, 12 (2019), 2254–2262
- Shiva Jahangiri. 2021. Wisconsin Benchmark Data Generator: To JSON and Beyond. In SIGMOD. ACM, 2887–2889

Contributions

- We define a scalable JSON document store benchmark with configurable, custom workloads, guided query and data generation.

Contributions

- We define a scalable JSON document store benchmark with configurable, custom workloads, guided query and data generation.

$$\begin{aligned} \texttt{SELECT}\ & f_1(x_1),\dots,f_n(x_n) \\ \texttt{FROM}\ & \texttt{"collection"} \\ \texttt{WHERE}\ & P_1(y_1)[\texttt{AND|OR}]\dots,P_n(y_m) \\ & [\texttt{ORDER BY|GROUP BY}]\ \ w_1\dots,w_k \end{aligned}$$

Contributions

- We define a scalable JSON document store benchmark with configurable, custom workloads, guided query and data generation.

$$\begin{aligned} \texttt{SELECT}\ & f_1(x_1),\ldots,f_n(x_n)\\ \texttt{FROM}\ & \texttt{"collection"}\\ \texttt{WHERE}\ & P_1(y_1)[\texttt{[AND|OR]}]\ldots,P_n(y_m)\\ & \texttt{[ORDER BY|GROUP BY]}\ \ w_1\ldots,w_k \end{aligned}$$

- We implement a fully functional DeepBench prototype for reproducible, comparable results across different systems.

Contributions

- We define a scalable JSON document store benchmark with configurable, custom workloads, guided query and data generation.

$$\begin{aligned}
\texttt{SELECT}\ & f_1(x_1), \ldots, f_n(x_n) \\
\texttt{FROM}\ & \texttt{"collection"} \\
\texttt{WHERE}\ & P_1(y_1)[\texttt{[AND|OR]}]\ldots, P_n(y_m) \\
& \texttt{[ORDER BY|GROUP BY]}\ \ w_1 \ldots, w_k
\end{aligned}$$

- We implement a fully functional DeepBench prototype for reproducible, comparable results across different systems.
- We conduct an experimental evaluation of several state-of-the-art document stores with DeepBench, resulting to several interesting insights that could not be found before.

Experiments

Conducted Experiments on JSON Document Stores

Setup:

- MongoDB (version 4.4.5) (mdb), PostgreSQL/JSON (version 13.2) (pqj)
- 120 cores (Intel® Xeron® CPU E7-4880 v2 @ 2.50Ghz), 500 GB DRAM
- DeepBench prototpye in Python

Experiments:

- Nested object (not shown), **array**
- OLAP by example of BI Benchmark

• Adrian Vogelsgesang, Michael Haubenschild, Jan Finis, Alfons Kemper, Viktor Leis, Tobias Mühlbauer, Thomas Neumann, Manuel Then: Get Real: How Benchmarks Fail to Represent the Real World. DBTest@SIGMOD 2018: 1:1-1:6)

UNNEST

UNNEST: function to return a result table with one row for each element of an array.

- $\{\texttt{"a"} : \begin{bmatrix}1 & 5 & -4 & 0\end{bmatrix}\} \xRightarrow{\texttt{UNNEST "a"}}$

"a"
1
5
-4
0

UNNEST

UNNEST: function to return a result table with one row for each element of an array.

- $\{\texttt{"a"} : \begin{bmatrix}1 & 5 & -4 & 0\end{bmatrix}\} \xRightarrow{\texttt{UNNEST "a"}}$

"a"
1
5
−4
0

- $\{\texttt{"b"} : \begin{bmatrix}\{\texttt{"a"}: \begin{bmatrix}1 & 2\end{bmatrix}\}, \\ \{\texttt{"a"}: \begin{bmatrix}3 & 4\end{bmatrix}\}\end{bmatrix}\} \xRightarrow{\texttt{UNNEST "b"} \rightarrow \texttt{UNNEST "a"}}$

"b"	"a"
$\{\texttt{"a"}: \begin{bmatrix}1 & 2\end{bmatrix}\}$	1
$\{\texttt{"a"}: \begin{bmatrix}1 & 2\end{bmatrix}\}$	2
$\{\texttt{"a"}: \begin{bmatrix}3 & 4\end{bmatrix}\}$	3
$\{\texttt{"a"}: \begin{bmatrix}3 & 4\end{bmatrix}\}$	4

UNNEST

UNNEST: function to return a result table with one row for each element of an array.

- $\{\texttt{"a"} : \begin{bmatrix}1 & 5 & -4 & 0\end{bmatrix}\} \xRightarrow{\texttt{UNNEST "a"}}$

"a"
1
5
−4
0

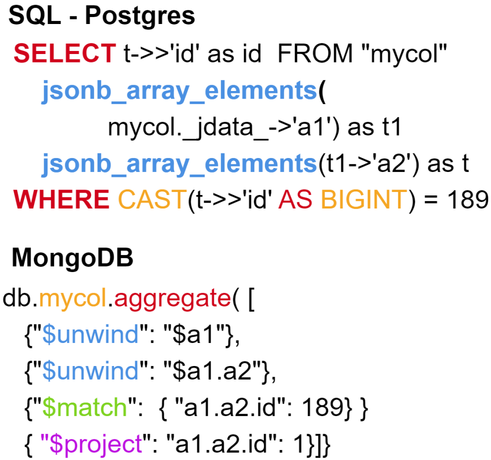
- $\{\texttt{"b"} : \begin{bmatrix}\{\texttt{"a"}: & \begin{bmatrix}1 & 2\end{bmatrix}\}, \\ \{\texttt{"a"}: & \begin{bmatrix}3 & 4\end{bmatrix}\}\end{bmatrix}\} \xRightarrow{\texttt{UNNEST "b"} \rightarrow \texttt{UNNEST "a"}}$

"b"	"a"
{"a": [1 2]}	1
{"a": [1 2]}	2
{"a": [3 4]}	3
{"a": [3 4]}	4

- Notation:

	Postgres	**MongoDB**
UNNEST	`jsonb_array_elements`	`$unwind`

UNNEST: Querying Nested JSON Arrays – Query



SQL - Postgres

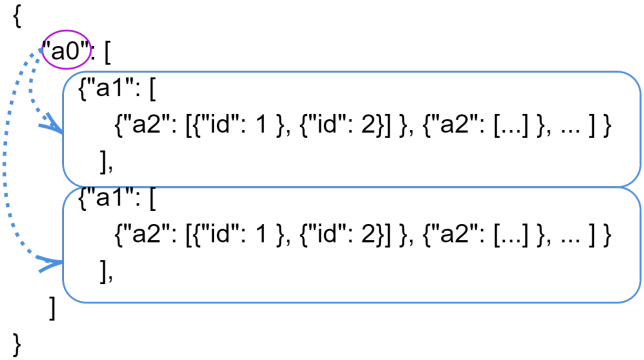
```
SELECT t->>'id' as id  FROM "mycol"
   jsonb_array_elements(
            mycol._jdata_->'a1') as t1
   jsonb_array_elements(t1->'a2') as t
WHERE CAST(t->>'id' AS BIGINT) = 189
```

MongoDB

```
db.mycol.aggregate( [
  {"$unwind": "$a1"},
  {"$unwind": "$a1.a2"},
  {"$match":  { "a1.a2.id": 189} }
  { "$project": "a1.a2.id": 1}]}
```

UNNEST: Querying Nested JSON Arrays – Data & Query

Document



```
{
  "a0": [
    {"a1": [
        {"a2": [{"id": 1 }, {"id": 2}] }, {"a2": [...] }, ... ] }
      ],
    {"a1": [
        {"a2": [{"id": 1 }, {"id": 2}] }, {"a2": [...] }, ... ] }
      ],
  ]
}
```

SQL - Postgres

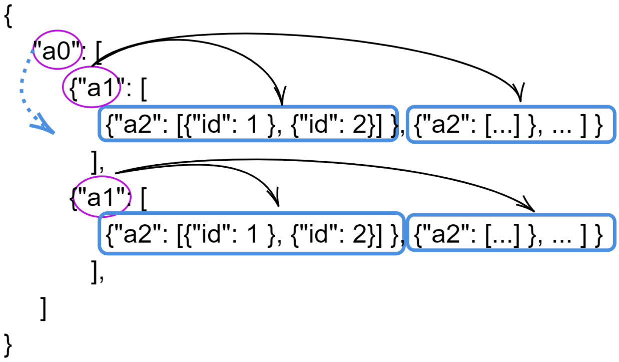
```
SELECT t->>'id' as id  FROM "mycol"
   jsonb_array_elements(
           mycol._jdata_->'a1') as t1
   jsonb_array_elements(t1->'a2') as t
WHERE CAST(t->>'id' AS BIGINT) = 189
```

MongoDB

```
db.mycol.aggregate( [
  {"$unwind": "$a1"},
  {"$unwind": "$a1.a2"},
  {"$match":  { "a1.a2.id": 189} }
  { "$project": "a1.a2.id": 1}]}
```

UNNEST: Querying Nested JSON Arrays – Data & Query

Document



```
{
  "a0": [
    {"a1": [
      {"a2": [{"id": 1 }, {"id": 2}] }, {"a2": [...] }, ... ] }
    ],
    {"a1": [
      {"a2": [{"id": 1 }, {"id": 2}] }, {"a2": [...] }, ... ] }
    ],
  ]
}
```

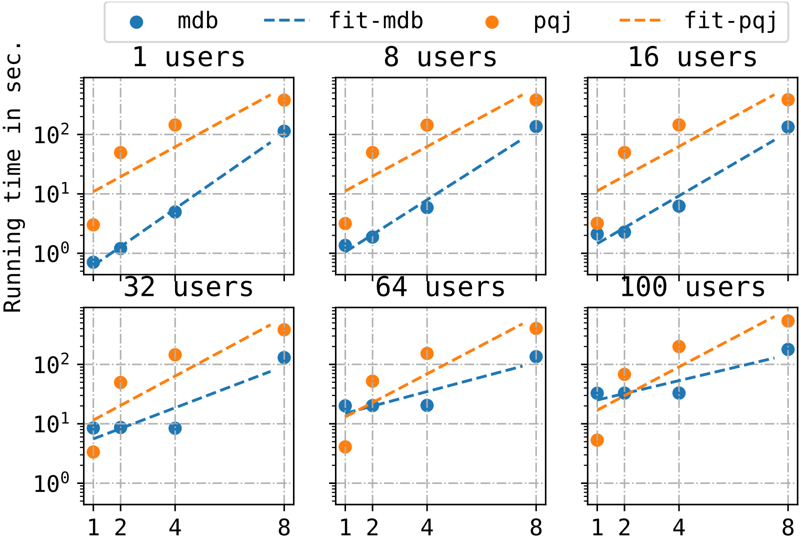
SQL - Postgres

```
SELECT t->>'id' as id  FROM "mycol"
   jsonb_array_elements(
            mycol._jdata_->'a1') as t1
   jsonb_array_elements(t1->'a2') as t
WHERE CAST(t->>'id' AS BIGINT) = 189
```

MongoDB

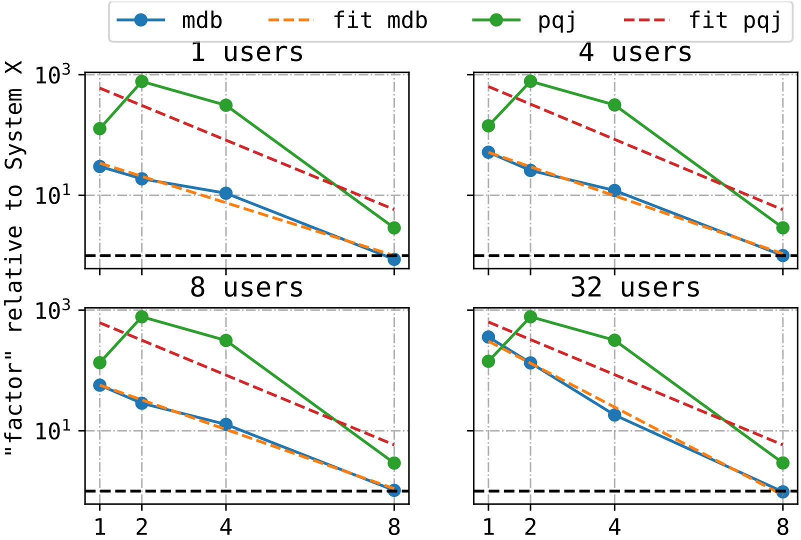
```
db.mycol.aggregate( [
  {"$unwind": "$a1"},
  {"$unwind": "$a1.a2"},
  {"$match":  { "a1.a2.id": 189} }
  { "$project": "a1.a2.id": 1}]}
```

UNNEST: Querying Nested JSON Arrays



on the x axis the number of sequential unnests

UNNEST: Querying Nested JSON Arrays



on the x axis the number of sequential UNNESTs

UNNEST: Querying Nested JSON Arrays – Conclusions

- Unnesting deeply nested arrays is a complex problem that shows a near exponential performance degradation for an increasing nesting level due to the high computational complexity of the similar array transposition problem;

UNNEST: Querying Nested JSON Arrays – Conclusions

- Unnesting deeply nested arrays is a complex problem that shows a near exponential performance degradation for an increasing nesting level due to the high computational complexity of the similar array transposition problem;
- Optimizations such as avoiding unnecessary unnest steps significantly improve the running times.

Custom Workloads: OLAP – Data & Query

- Example document from BI benchmark workbook `Food`

```
{ "Number_of_Records": 1,      "activity_sec": 20,
  "application": "Blogger",    "device": "6681",
  "volume_total_bytes": 7604, "subscribers": 3    }
```

- Adrian Vogelsgesang, Michael Haubenschild, Jan Finis, Alfons Kemper, Viktor Leis, Tobias Mühlbauer, Thomas Neumann, and Manuel Then. 2018. Get Real: How Benchmarks Fail to Represent the Real World. In DBTest@SIGMOD, Alexan- der Böhm and Tilmann Rabl (Eds.). ACM, 1:1–1:6.

Custom Workloads: OLAP – Data & Query

- Example document from BI benchmark workbook `Food`

```
{ "Number_of_Records": 1,      "activity_sec": 20,
  "application": "Blogger",    "device": "6681",
  "volume_total_bytes": 7604,  "subscribers": 3    }
```

- Query from original workbook

```
SELECT
  "Food_1"._jdata_.->>'device" AS "device'
FROM "Food_1"
GROUP BY "Food_1"._jdata_->>'device'ORDER BY "device" ASC;
```

• Adrian Vogelsgesang, Michael Haubenschild, Jan Finis, Alfons Kemper, Viktor Leis, Tobias Mühlbauer, Thomas Neumann, and Manuel Then. 2018. Get Real: How Benchmarks Fail to Represent the Real World. In DBTest@SIGMOD, Alexan- der Böhm and Tilmann Rabl (Eds.). ACM, 1:1–1:6.

Custom Workloads: OLAP – Data & Query

- Example document from BI benchmark workbook `Food`

```
{ "Number_of_Records": 1,      "activity_sec": 20,
  "application": "Blogger",    "device": "6681",
  "volume_total_bytes": 7604, "subscribers": 3   }
```

- Query from original workbook

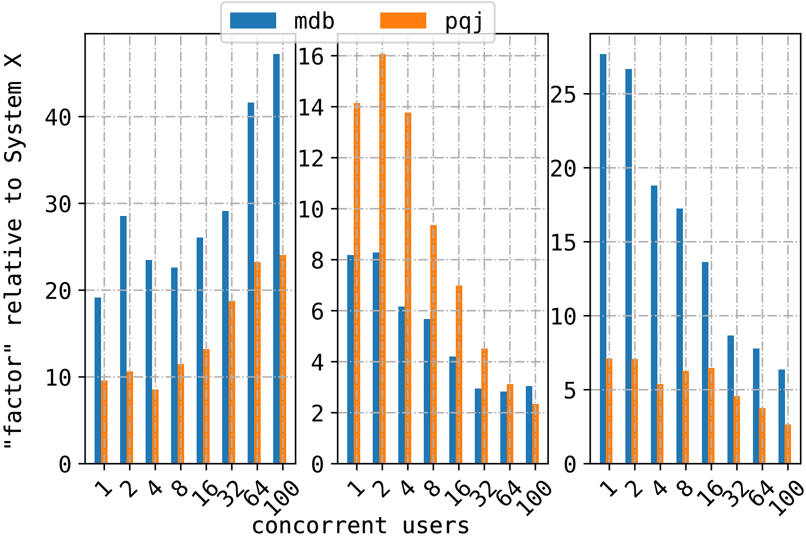
```
SELECT
  "Food_1"._jdata_.->>'device" AS "device'
FROM "Food_1"
GROUP BY "Food_1"._jdata_->>'device'ORDER BY "device" ASC;
```

- Automatically generated query

```
SELECT
   AVG(CAST("Food_1"._jdata_.->>'volume_total_bytes' AS DOUBLE))
FROM "Food_1"
WHERE CAST("Food_1"._jdata_->>'activity_sec' AS BIGINT) = 20;
```

• Adrian Vogelsgesang, Michael Haubenschild, Jan Finis, Alfons Kemper, Viktor Leis, Tobias Mühlbauer, Thomas Neumann, and Manuel Then. 2018. Get Real: How Benchmarks Fail to Represent the Real World. In DBTest@SIGMOD, Alexan- der Böhm and Tilmann Rabl (Eds.). ACM, 1:1–1:6.

Custom Workloads: OLAP – BI Benchmark



GROUP BY and ORDER BY without index (left), AVG on filtered set with index (center), and without (right)

Custom Workloads: BI Benchmark – Conclusions

- There seems to be performance improvement potential for document stores in the context of analytical query processing;

Custom Workloads: BI Benchmark – Conclusions

- There seems to be performance improvement potential for document stores in the context of analytical query processing;
- Improvements like indexing and analytical processing techniques from column stores are beneficial and could be transferred to document store

Conclusions

Conclusions

- This work addresses an important shortcoming in document store benchmarking, namely the lack of JSON-specific data and query generation.

Conclusions

- This work addresses an important shortcoming in document store benchmarking, namely the lack of JSON-specific data and query generation.
- For that we propose, DeepBench, an extensible, scalable, JSON-specific benchmark, with which we evaluated two well-known document stores.

Conclusions

- This work addresses an important shortcoming in document store benchmarking, namely the lack of JSON-specific data and query generation.
- For that we propose, DeepBench, an extensible, scalable, JSON-specific benchmark, with which we evaluated two well-known document stores.
- For the time, it is possible to gain deeper insights into strengths, weaknesses of these systems, especially in the areas of nested arrays and analytical queries, and potential improvements

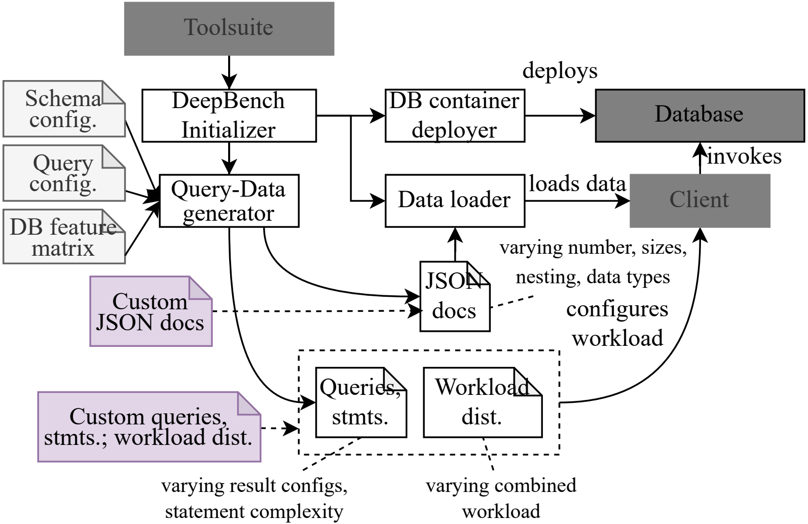
Conclusions

- This work addresses an important shortcoming in document store benchmarking, namely the lack of JSON-specific data and query generation.
- For that we propose, DeepBench, an extensible, scalable, JSON-specific benchmark, with which we evaluated two well-known document stores.
- For the time, it is possible to gain deeper insights into strengths, weaknesses of these systems, especially in the areas of nested arrays and analytical queries, and potential improvements
- In future work, we plan to further investigate the identified weaknesses and explore solutions.

Thank You for Your Attention!

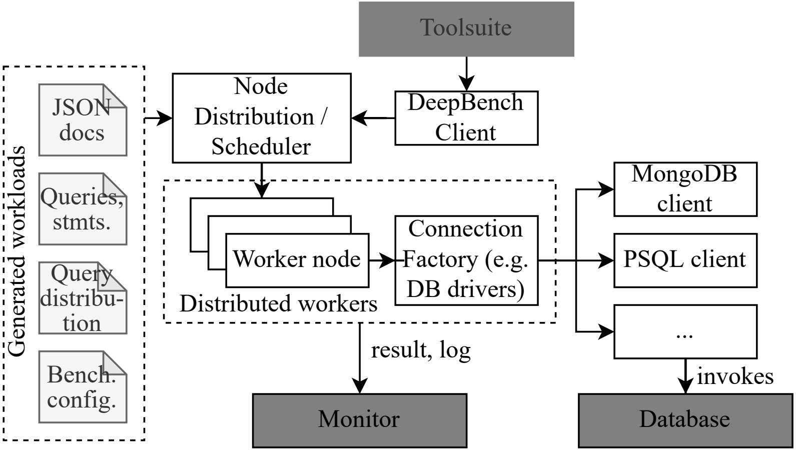
Contact: Stefano Belloni (stefano.belloni@sap.com),
Daniel Ritter (daniel.ritter@sap.com)

DeepBench initializer

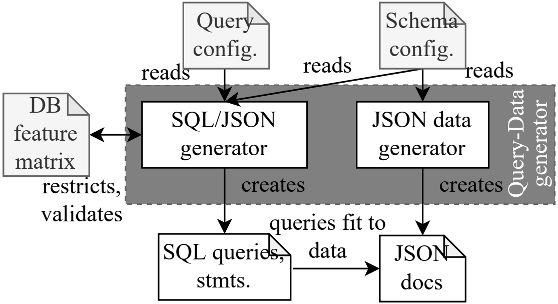


- query and data generation

DeepBench

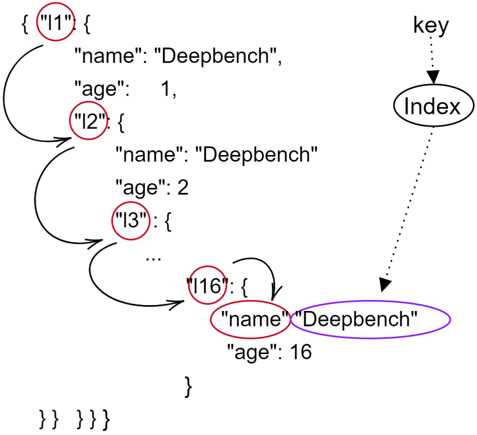

Toolsuite
Generated workloads
JSON docs
Queries, stmts.
Query distribu- tion
Bench. config.
Node Distribution / Scheduler
DeepBench Client
Worker node
Distributed workers
Connection Factory (e.g. DB drivers)
MongoDB client
PSQL client
...
result, log
invokes
Monitor
Database

Query Generation



Deep: Querying Nested JSON Objects – Data & Query

Document



```
{ "l1": {
      "name": "Deepbench",
      "age":    1,
      "l2": {
            "name": "Deepbench"
            "age": 2
            "l3": {
                  ...
                        "l16": {
                              "name""Deepbench"
                              "age": 16
                        }
}}  }}}
```

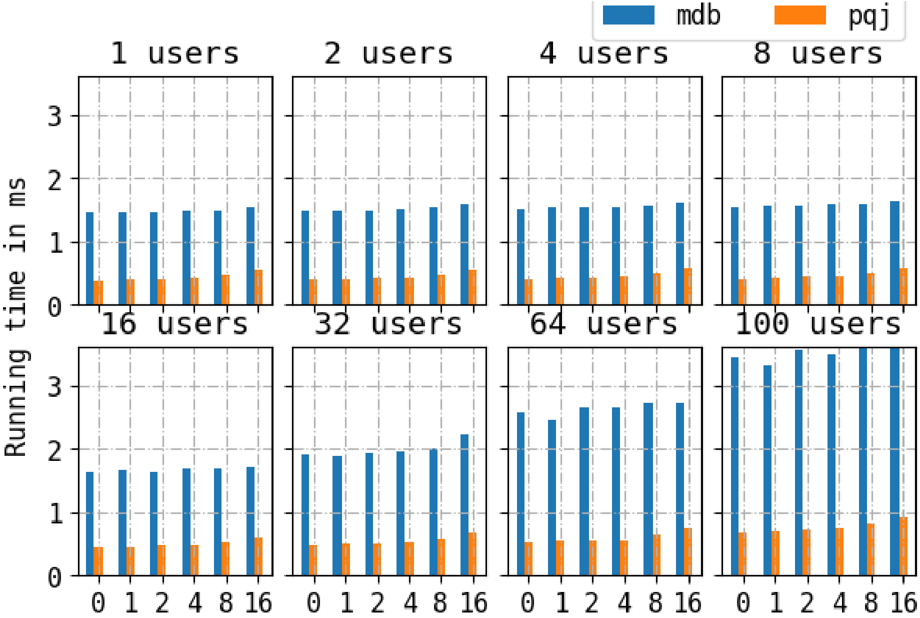
SQL-Postgres

```
SELECT _JDATA_->>'name'
 FROM "deepbench"
WHERE
      _JDATA_->'l1'
              ->'l2' ...
           ...  -> 'l16'->>'name' = 'Deepbench'
```

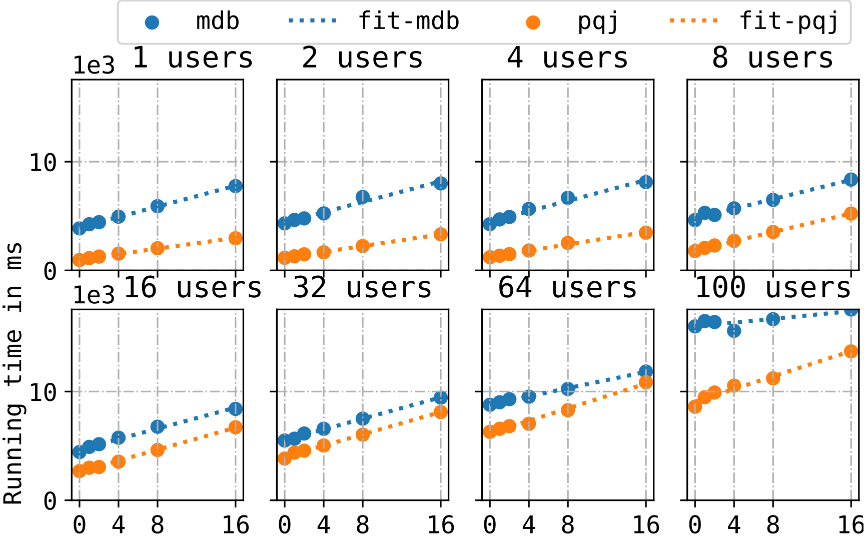
MongoDB

```
db.deepbanch.find(
    {'l1'.'l2'. ... . l16.name': Deepbench},
    {'_id': 0, 'name': 1}
)
```

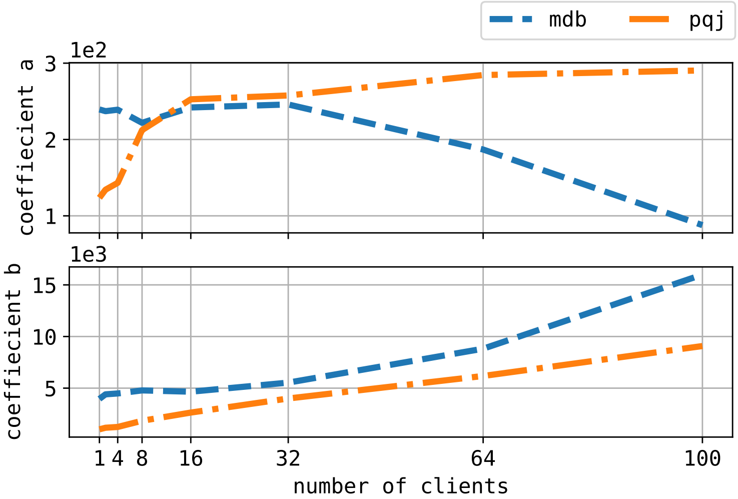
Deep: Querying Nested JSON Objects – Using an Index



Deep: Querying Nested JSON Objects – without an Index



Deep: Querying Nested JSON Objects – Linear regression



Linear Regression: $t = a \cdot \texttt{level} + b$

Deep: Querying Nested JSON Objects – Conclusions

- The *depth* dimension of document object access is clearly an important aspect that can have negative impact on the performance;

Deep: Querying Nested JSON Objects – Conclusions

- The *depth* dimension of document object access is clearly an important aspect that can have negative impact on the performance;
- This degradation becomes more and more negligible with the increase of concurrent users and in particular is completely eliminated when an index is used.