Performance Evaluation of MongoDB I/O access patterns

Abdulqawi Saif\textsuperscript{1,2,3,4}  
abdulqawi.saif@loria.fr

Lucas Nussbaum\textsuperscript{1,2,3}  
lucas.nussbaum@loria.fr

Ye-Qiong Song\textsuperscript{1,2,3}  
ye-qion.song@loria.fr

\textsuperscript{1}Inria, Villers-les-Nancy, F-54600, France  
\textsuperscript{2}Université de Lorraine, LORIA, F-54500, France  
\textsuperscript{3} CNRS, LORIA - UMR 7503, F-54500, France  
\textsuperscript{4}Xilopix SAS, F-88000, France

September 07, 2017  
CLOUD DAYS’2017 - Nancy, France
Context

NoSQL databases

- Principle category of Big Data Storage
- Overcome the scalability issues of SQL DBs
- Almost Cloud-friendly systems
- Different architectures and data-models

MongoDB

- A document-based NoSQL database (JSON documents)
- It uses *sharding* for distributing data
- Different tools for easy-deployment and migration (Ex. mongorestore, mongodump)
- Simple to use and fit well with various kinds of apps.
Context

Querying data

- NoSQL flexibility: store data; think how to use after!
- No schema $\Rightarrow$ not only one way to query the data

frequent need to create indexes on demand, on large-scale data!
Querying data

- NoSQL flexibility: store data; think how to use after!
- No schema ⇒ not only one way to query the data

frequent need to create indexes on demand, on large-scale data!

Problem statement

- MongoDB takes unjustified amount of time for making indexes on a pre-existed data
  
  this problem could affect other NoSQL databases!
Benchmarking?

⇒ Tools such as *Yahoo! Cloud Serving Benchmarks (YCSB)* could:
  - report high level metrics (Ex. throughput, latency, ...) ✓
  - reduce the effort for comparing different systems ✓

But

- could not reflect the complexity of production-like data 😞
- are not suitable for explaining performance issues 😞

Research objectives

- Identifying the performance issue(s) of MongoDB while indexing data
- Introducing new experimental tools for explaining the causes of performance issue(s)
Experiments

Datasets

<table>
<thead>
<tr>
<th>Dataset type</th>
<th>(min, avrg, max) in Kb</th>
<th>N. of data units</th>
<th>Size (Gb)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDU₁</td>
<td>(1, 3, 6)</td>
<td>20,000,000 docs</td>
<td>71</td>
</tr>
</tbody>
</table>

- Every doc x (int, date, 2 x string[min, max], array[1..4] x string[min, max])

MongoDB’s storage engine: Wiredtiger

- The default storage engine for persistent data.
- _id’s for holding the collection’s records
- collection is a separate file
- Btree data structure on disks
- Contiguous allocation on disk, but unspanned
- Default page size is 4 KB

Figure – a) Spanned & b) Unspanned allocation
Experiments

Experimental setup

- Experiments are performed on Grid’5000 testbed
  - Machines x 2 CPUs Intel Xeon E5-2630 v3, 8 cores/CPU, 128GB RAM, 2x558GB HDD, 10Gbps ethernet
- Ubuntu 14.04, Linux 4.9.13, MongoDB v3.4
- wiredTiger cache = 2Gb, replicas are disabled, sharding key is \_id
- Considering only hash sharding in this talk ⇒ load balancing

Figure – Experimental scenarios
Experiments: Performance results with $SDU_1$

Performance Evaluation of MongoDB I/O access patterns
Experiments: Performance results with $SDU_1$

⇒ Indexing is a 100% read workload
⇒ I/O bounded operations
⇒ Data distribution issue
⇒ Benchmarking is not sufficient to look beyond!
Experiments: Performance results with $SDU_1$

- **Standalone MongoDB**
  - Execution time (secs)
  - Throughput (MBps)
  - Read
  - Write

- **Distributed MongoDB**
  - Execution time (secs)
  - Throughput (MBps)
  - Shard1 (50.9% of data)
  - Shard2 (49.1% of data)

⇒ Indexing is a 100% read workload
⇒ I/O bounded operations
⇒ Data distribution issue
⇒ Benchmarking is not sufficient to look beyond!
Experiments: Performance results with $SDU_1$

- Indexing is a 100% read workload
- I/O bounded operations
- Data distribution issue
- Benchmarking is not sufficient to look beyond!
Experiments: Performance results with $SDU_1$

- **Standalone MongoDB**
  - Execution time (secs)
  - Throughput (MBps)
  - Utilization (%)

- **Distributed MongoDB**
  - Execution time (secs)
  - Throughput (MBps)
  - Utilization (%)

- **Shard1 (50.9% of data)**
- **shard2 (49.1% of data)**

- **Doubling the time!**
- **0 gain!**

- **⇒ Indexing is a 100% read workload**
- **⇒ I/O bounded operations**
- **⇒ Data distribution issue**
- **⇒ Benchmarking is not sufficient to look beyond!**
Experiments : Performance results with $SDU_1$

⇒ Indexing is a 100% read workload
⇒ I/O bounded operations
⇒ Data distribution issue
⇒ Benchmarking is not sufficient to look beyond!

⇒ How to get the main reason behind these results?
## Dynamic tracing with eBPF

### extended Berkeley Packet Filter (eBPF)

- Connect to all Linux data sources: (Kprobes, Uprobes, tracepoints, ...)
- Almost no overhead (4 ns per syscall) [A. Starovoitov]
- In-kernel verifier, JIT-compilation, mapping (kernel-userspace exchange)
- Processing, tracing, filtering inside Linux kernel.
- Suitable to work with system in-production & real data.

### BPF compiler collection (BCC)

- Like-standard frontend project for eBPF tools
- No more byte code! ⇒ write your BPF code in restricted C
- Write your frontend code using Python or Lua
- Towards script-driven tracing
Dynamic tracing with eBPF, our tool

An eBPF tool is built to evaluate the I/O patterns

- I/O access are no longer considered as black boxes!
- A generic tool, works on VFS layer
- It traces all I/O read requests
- It reports all files offsets, req. latencies & data size
- Could filter the results by file

Tracing impact on our experiments

- Overhead cost is lower than 0.8% of execution time in all Experiments.
- Basically, it depends on the number issued I/O requests.
Experiments: eBPF results with $SDU_1$

I/O access pattern and latency on standalone config.
Experiments: eBPF results with $SDU_1$

I/O access pattern and latency on standalone config.
Experiments: eBPF results with $SDU_1$

⇒ Acceptable access pattern on standalone
⇒ Almost random access on shards!
⇒ Data distribution is poorly done

First shard results in two-shards config (50.9% of data)

Second shard results in two-shards config (49.1% of data)
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection _ids:
used by MongoDB process and WiredTiger

<table>
<thead>
<tr>
<th>_id 1</th>
<th>_id 2</th>
<th>_id 3</th>
<th>_id 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>_id 7</td>
<td>_id 8</td>
<td>_id 10</td>
<td>_id 11</td>
</tr>
</tbody>
</table>
|  _id 19 | _id 20 | _id 27 |   ...

Collection file on disk:
allocated regarding the key sharding (hashed _id)

<table>
<thead>
<tr>
<th>_id 1</th>
<th>_id 3</th>
<th>_id 7</th>
<th>_id 5</th>
<th>_id 8</th>
<th>_id 10</th>
<th>_id 2</th>
<th>_id 19</th>
<th>_id 11</th>
<th>_id 20</th>
<th>_id 30</th>
<th>_id 27</th>
</tr>
</thead>
<tbody>
<tr>
<td>rec 1</td>
<td>rec 2</td>
<td>rec 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>rec(n-1)</td>
<td>rec(n)</td>
</tr>
</tbody>
</table>

**Figure** – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection _ids:
used by MongoDB process and WiredTiger

<table>
<thead>
<tr>
<th>_id 1</th>
<th>_id 2</th>
<th>_id 3</th>
<th>_id 5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_id 7</td>
<td>_id 8</td>
<td>_id 10</td>
<td>_id 11</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>_id 19</td>
<td>_id 20</td>
<td>_id 27</td>
<td>...</td>
</tr>
</tbody>
</table>

1) Get this doc

Collection file on disk:
allocation regarding the key sharding (hashed _id)

<table>
<thead>
<tr>
<th>_id 1</th>
<th>_id 3</th>
<th>_id 7</th>
<th>_id 5</th>
<th>id 8</th>
<th>_id 10</th>
<th>_id 2</th>
<th>_id 19</th>
<th>_id 11</th>
<th>_id 20</th>
<th>_id 30</th>
<th>_id 27</th>
</tr>
</thead>
<tbody>
<tr>
<td>rec 1</td>
<td>rec 2</td>
<td>rec 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure** – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection _ids:
used by MongoDB process and WiredTiger

```
_id 1  _id 2  _id 3  _id 5
_id 7  _id 8  _id 10 _id 11
_id 19 _id 20 _id 27 ...
```

1) Get this doc

Collection file on disk:
allocated regarding the key sharding (hashed _id)

```
_id 1  _id 3  _id 7  _id 5  _id 8  _id 10 _id 2  _id 19 _id 11 _id 20 _id 30 _id 27
rec 1  rec 2  rec 3  
```

**Figure** – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection _ids:
used by MongoDB process and WiredTiger

<table>
<thead>
<tr>
<th>_id 1</th>
<th>_id 2</th>
<th>_id 3</th>
<th>_id 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>_id 7</td>
<td>_id 8</td>
<td>_id 10</td>
<td>_id 11</td>
</tr>
<tr>
<td>_id 19</td>
<td>_id 20</td>
<td>_id 27</td>
<td>...</td>
</tr>
</tbody>
</table>

1) Get this doc
2) Which doc is next?

Collection file on disk:
allocated regarding the key sharding (hashed _id)

<table>
<thead>
<tr>
<th>_id 1</th>
<th>_id 3</th>
<th>_id 7</th>
<th>_id 5</th>
<th>_id 8</th>
<th>_id 10</th>
<th>_id 2</th>
<th>_id 19</th>
<th>_id 11</th>
<th>_id 20</th>
<th>_id 30</th>
<th>_id 27</th>
</tr>
</thead>
<tbody>
<tr>
<td>rec 1</td>
<td>rec 2</td>
<td>rec 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>rec(n-1)</td>
<td>rec (n)</td>
</tr>
</tbody>
</table>

**Figure** – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection _ids:
used by MongoDB process and WiredTiger

<table>
<thead>
<tr>
<th>_id 1</th>
<th>_id 2</th>
<th>_id 3</th>
<th>_id 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>_id 7</td>
<td>_id 8</td>
<td>_id 10</td>
<td>_id 11</td>
</tr>
<tr>
<td>_id 19</td>
<td>_id 20</td>
<td>_id 27</td>
<td>...</td>
</tr>
</tbody>
</table>

1) Get this doc
2) Which doc is next?

Collection file on disk:
allocated regarding the key sharding (hashed _id)

<table>
<thead>
<tr>
<th>_id 1</th>
<th>_id 3</th>
<th>_id 7</th>
<th>_id 5</th>
<th>_id 8</th>
<th>_id 10</th>
<th>_id 2</th>
<th>_id 19</th>
<th>_id 11</th>
<th>_id 20</th>
<th>_id 30</th>
<th>_id 27</th>
</tr>
</thead>
<tbody>
<tr>
<td>rec 1</td>
<td>rec 2</td>
<td>rec 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>rec(n-1)</td>
<td>rec (n)</td>
</tr>
</tbody>
</table>

**Figure** – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

![Diagram showing the mismatch between the scanning table and data stored on disk. The figure illustrates the collection _ids used by MongoDB process and WiredTiger, and the collection file on disk allocated regarding the key sharding (hashed _id).]

**Figure** – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection _ids:
used by MongoDB process and WiredTiger

<table>
<thead>
<tr>
<th>_id 1</th>
<th>_id 2</th>
<th>_id 3</th>
<th>_id 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>_id 7</td>
<td>_id 8</td>
<td>_id 10</td>
<td>_id 11</td>
</tr>
<tr>
<td>_id 19</td>
<td>_id 20</td>
<td>_id 27</td>
<td>...</td>
</tr>
</tbody>
</table>

1) Get this doc
2) Which doc is next?

Collection file on disk:
allocated regarding the key sharding (hashed _id)

<table>
<thead>
<tr>
<th>_id 1</th>
<th>_id 3</th>
<th>_id 7</th>
<th>_id 5</th>
<th>_id 8</th>
<th>_id 10</th>
<th>_id 2</th>
<th>_id 19</th>
<th>_id 11</th>
<th>_id 20</th>
<th>_id 30</th>
<th>_id 27</th>
</tr>
</thead>
<tbody>
<tr>
<td>rec 1</td>
<td>rec 2</td>
<td>rec 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>rec(n-1)</td>
</tr>
</tbody>
</table>

**Figure** — MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection _ids:
used by MongoDB process and WiredTiger

1) Get this doc

Collection file on disk:
allocated regarding the key sharding (hashed _id)

**Figure** – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection _ids:
used by MongoDB process and WiredTiger

<table>
<thead>
<tr>
<th>_id 7</th>
<th>_id 8</th>
<th>_id 10</th>
<th>_id 11</th>
</tr>
</thead>
<tbody>
<tr>
<td>_id 19</td>
<td>_id 20</td>
<td>_id 27</td>
<td>...</td>
</tr>
</tbody>
</table>

1) Get this doc
2) Which doc is next?

Collection file on disk:
allocated regarding the key sharding (hashed _id)

<table>
<thead>
<tr>
<th>_id 1</th>
<th>_id 3</th>
<th>_id 7</th>
<th>_id 5</th>
<th>_id 8</th>
<th>_id 10</th>
<th>_id 2</th>
<th>_id 19</th>
<th>_id 11</th>
<th>_id 20</th>
<th>_id 30</th>
<th>_id 27</th>
</tr>
</thead>
<tbody>
<tr>
<td>rec 1</td>
<td>rec 2</td>
<td>rec 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure** – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection _ids:
used by MongoDB process and WiredTiger

Collection file on disk:
allocated regarding the key sharding (hashed _id)

**Figure** – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection _ids:
used by MongoDB process and WiredTiger

Collection file on disk:
allocated regarding the key sharding (hashed _id)

1) Get this doc
2) Which doc is next?

**Figure** – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection _ids:
used by MongoDB process and WiredTiger

1) Get this doc

Collection file on disk:
allocated regarding the key sharding (hashed _id)

**Figure** – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection _ids:
used by MongoDB process and WiredTiger

<table>
<thead>
<tr>
<th>_id 1</th>
<th>_id 2</th>
<th>_id 3</th>
<th>_id 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>_id 7</td>
<td>_id 8</td>
<td>_id 10</td>
<td>_id 11</td>
</tr>
<tr>
<td>_id 19</td>
<td>_id 20</td>
<td>_id 27</td>
<td>...</td>
</tr>
</tbody>
</table>

1) Get this doc
2) Which doc is next?

Collection file on disk:
allocated regarding the key sharding (hashed _id)

<table>
<thead>
<tr>
<th>_id 1</th>
<th>_id 3</th>
<th>_id 7</th>
<th>_id 5</th>
<th>id 8</th>
<th>_id 10</th>
<th>_id 2</th>
<th>_id 19</th>
<th>id 11</th>
<th>id 20</th>
<th>_id 30</th>
<th>_id 27</th>
</tr>
</thead>
<tbody>
<tr>
<td>rec 1</td>
<td>rec 2</td>
<td>rec 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>rec(n-1)</td>
<td>rec (n)</td>
</tr>
</tbody>
</table>

**Figure** – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- **Mismatch between the scanning table vs data stored on disk**

Collection _ids:
used by MongoDB process and WiredTiger

1) Get this doc

Collection file on disk:
allocated regarding the key sharding (hashed _id)

**Figure** – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection _ids:
used by MongoDB process and WiredTiger

<table>
<thead>
<tr>
<th>_id 1</th>
<th>_id 2</th>
<th>_id 3</th>
<th>_id 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>_id 7</td>
<td>_id 8</td>
<td>_id 10</td>
<td>_id 11</td>
</tr>
<tr>
<td>_id 19</td>
<td>_id 20</td>
<td>_id 27</td>
<td>...</td>
</tr>
</tbody>
</table>

1) Get this doc
2) Which doc is next?

Collection file on disk:
allocated regarding the key sharding (hashed _id)

<table>
<thead>
<tr>
<th>_id 1</th>
<th>_id 3</th>
<th>_id 7</th>
<th>_id 5</th>
<th>_id 8</th>
<th>_id 10</th>
<th>_id 2</th>
<th>_id 19</th>
<th>_id 11</th>
<th>_id 20</th>
<th>_id 30</th>
<th>_id 27</th>
</tr>
</thead>
<tbody>
<tr>
<td>rec 1</td>
<td>rec 2</td>
<td>rec 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure** – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

**Figure** – MongoDB scanning table Vs records order on the disk

Collection `_ids`:
used by MongoDB process and WiredTiger

1) Get this doc

Collection file on disk:
allocated regarding the key sharding (hashed `_id`)

**Figure** – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection _ids:
used by MongoDB process and WiredTiger

<table>
<thead>
<tr>
<th>_id 1</th>
<th>_id 2</th>
<th>_id 3</th>
<th>_id 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>_id 7</td>
<td>_id 8</td>
<td>_id 10</td>
<td>_id 11</td>
</tr>
<tr>
<td>_id 19</td>
<td>_id 20</td>
<td>_id 27</td>
<td>...</td>
</tr>
</tbody>
</table>

1) Get this doc
2) Which doc is next?

Collection file on disk:
allocated regarding the key sharding (hashed _id)

<table>
<thead>
<tr>
<th>_id 1</th>
<th>_id 3</th>
<th>_id 7</th>
<th>_id 5</th>
<th>_id 8</th>
<th>_id 10</th>
<th>_id 2</th>
<th>_id 19</th>
<th>_id 11</th>
<th>_id 20</th>
<th>_id 30</th>
<th>_id 27</th>
</tr>
</thead>
<tbody>
<tr>
<td>rec 1</td>
<td>rec 2</td>
<td>rec 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure** – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection `_ids`:
used by MongoDB process and WiredTiger

---

Collection file on disk:
allocated regarding the key sharding (hashed `_id`)

---

**Figure** – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection `_id`s:
used by MongoDB process and WiredTiger

```
_id 1  _id 2  _id 3  _id 5
_id 7  _id 8  _id 10 _id 11
_id 19 _id 20 _id 27 ...
```

1) Get this doc
2) Which doc is next?

Collection file on disk:
allocated regarding the key sharding (hashed `_id`)

```
_rec 1  _id 3  _id 7  _id 5  _id 8  _id 10 _id 2  _id 19 _id 11 _id 20 _id 30 _id 27
_rec 2  _rec 3  _rec 1  _rec 2  ...
```

Figure – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection _ids:
used by MongoDB process and WiredTiger

```
_id 1  _id 2  _id 3  _id 5
_id 7  _id 8  _id 10 _id 11
_id 19 _id 20 _id 27 ..
```

1) Get this doc

Collection file on disk:
allocated regarding the key sharding (hashed _id)

```
_id 1  _id 3  _id 7  _id 5  _id 8  _id 10 _id 2  _id 19 _id 11 _id 20 _id 30 _id 27
rec 1  rec 2  rec 3 _id 5  _id 8  _id 10 _id 2  _id 19 _id 11 _id 20 _id 30 _id 27
```

Figure – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection _ids:
used by MongoDB process and WiredTiger

<table>
<thead>
<tr>
<th>_id 1</th>
<th>_id 2</th>
<th>_id 3</th>
<th>_id 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>_id 7</td>
<td>_id 8</td>
<td>_id 10</td>
<td>_id 11</td>
</tr>
<tr>
<td>_id 19</td>
<td>_id 20</td>
<td>_id 27</td>
<td>...</td>
</tr>
</tbody>
</table>

1) Get this doc
2) Which doc is next?

Collection file on disk:
allocated regarding the key sharding (hashed _id)

<table>
<thead>
<tr>
<th>_id 1</th>
<th>_id 3</th>
<th>_id 7</th>
<th>_id 5</th>
<th>_id 8</th>
<th>_id 10</th>
<th>_id 2</th>
<th>_id 19</th>
<th>_id 11</th>
<th>_id 20</th>
<th>_id 30</th>
<th>_id 27</th>
</tr>
</thead>
<tbody>
<tr>
<td>rec 1</td>
<td>rec 2</td>
<td>rec 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure** – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection _ids:
used by MongoDB process and WiredTiger

```
_id 1  _id 2  _id 3  _id 5
_id 7  _id 8  _id 10 _id 11
_id 19 _id 20 _id 27 ...
```

1) Get this doc

Collection file on disk:
allocated regarding the key sharding (hashed _id)

```
_id 1  _id 3  _id 7  _id 5  _id 8  _id 10 _id 2  _id 19 _id 11 _id 20
rec 1  rec 2  rec 3  _id 30 _id 27
```

**Figure** – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection _ids:
used by MongoDB process and WiredTiger

1) Get this doc

Collection file on disk:
allocated regarding the key sharding (hashed _id)

Figure – MongoDB scanning table Vs records order on the disk
Experiments

The exact issue

- Mismatch between the scanning table vs data stored on disk

Collection _ids: used by MongoDB process and WiredTiger

1) Get this doc
2) Which doc is next?

Collection file on disk: allocated regarding the key sharding (hashed _id)

**Figure** – MongoDB scanning table Vs records order on the disk
Experiments: an ad hoc solution

<table>
<thead>
<tr>
<th>MongoDB integrated tools?</th>
</tr>
</thead>
<tbody>
<tr>
<td>- RepairDatabase ⇒ verifying the coherence of data. 😞</td>
</tr>
<tr>
<td>- Rebuilding the _id index ⇒ consider the pre-existent ordering. 😞</td>
</tr>
</tbody>
</table>
Experiments: an ad hoc solution

MongoDB integrated tools?

- `RepairDatabase` ⇒ verifying the coherence of data. 😞
- `Rebuilding the `_id` index` ⇒ consider the pre-existing ordering. 😞

Our ad-hoc solution

- recreating the `_id` index regarding how the data is stored on disk.
Experiments: an ad hoc solution

MongoDB integrated tools?
- RepairDatabase ⇒ verifying the coherence of data. 😞
- Rebuilding the _id index ⇒ consider the pre-existed ordering. 😞

Our ad-hoc solution
- recreating the _id index regarding how the data is stored on disk.

The worst case example!
- Data on one shard in a four conf. experiment (1/4 out of data)
- indexing time: Doubling the indexing time of the whole data on standalone config (3000 sec).
- indexing time after: six time faster! (about 500 sec)
Experiments: an ad hoc solution

MongoDB integrated tools?

- RepairDatabase ⇒ verifying the coherence of data.
- Rebuilding the _id index ⇒ consider the pre-existed ordering.

Our ad-hoc solution

- recreating the _id index regarding how the data is stored on disk.

The worst case example!

- Data on one shard in a four conf. experiment (1/4 out of data)
- indexing time: Doubling the indexing time of the whole data on standalone config (3000 sec).
- indexing time after: six time faster! (about 500 sec)

**Figure** – I/O access pattern a) before & b) after applying the solution
Conclusion

- A performance study on MongoDB I/O access pattern is done
- A generic tool for testing I/O access patterns is introduced
- An ad hoc solution is proposed to correct MongoDB I/O access patterns

- We showed how a trivial data management issue could affect the performance
- We demonstrated how it is worthy to use dynamic tracing to go beyond benchmarking results

Future works

- Including other solutions like Cassandra in those tests.
- Trying to enrich eBPF tools which explain specific issues.
Conclusion

- A performance study on MongoDB I/O access pattern is done
- A generic tool for testing I/O access patterns is introduced
- An ad hoc solution is proposed to correct MongoDB I/O access patterns

- We showed how a trivial data management issue could affect the performance
- We demonstrated how it is worthy to use dynamic tracing to go beyond benchmarking results

Future works

- Including other solutions like Cassandra in those tests.
- Trying to enrich eBPF tools which explain specific issues.

Questions are welcome!