Performing Low-Level I/O Evaluations for Discovering Potential I/O Issues using IOscope

Abdulqawi Saif[1,2]
abdulqawi.saif@loria.fr

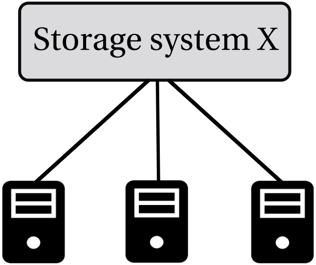
Lucas Nussbaum[1]
lucas.nussbaum@loria.fr

Ye-Qiong Song[1]
ye-qiong.song@loria.fr

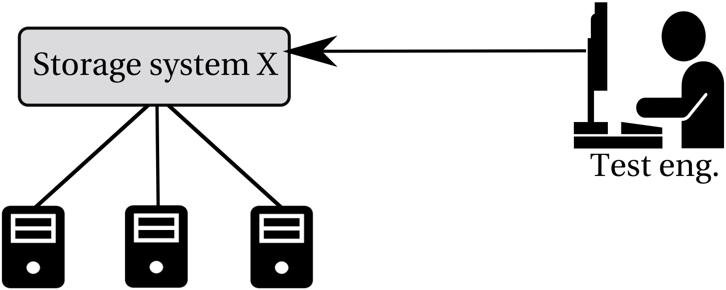
[1]Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy, France
[2]Qwant Enterprise, F-88000 Épinal, France

January 25, 2019
Per3S 2019 – Performance and Scalability of Storage Systems
Bordeaux, France

How is I/O performance often evaluated?

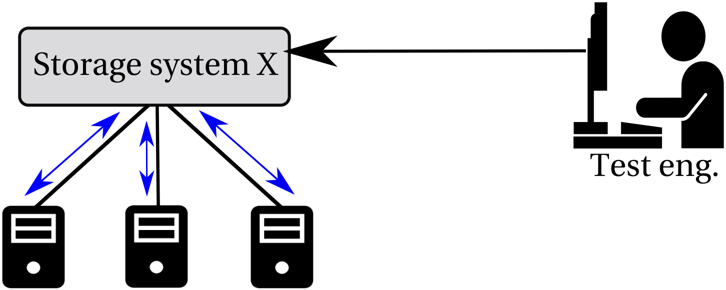


How is I/O performance often evaluated?



⇒ Communication through high-level interfaces (REST, API, . . .)

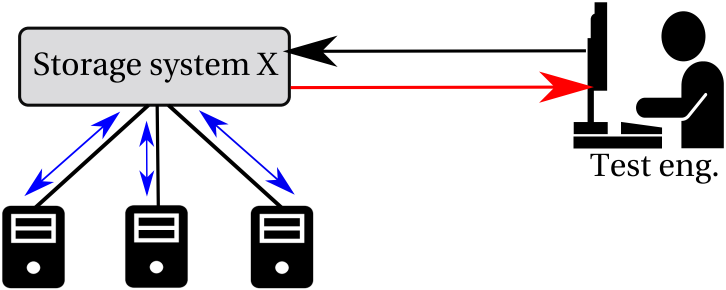
How is I/O performance often evaluated?



⇒ Workload execution phase

⇒ Configuration determines which nodes to use

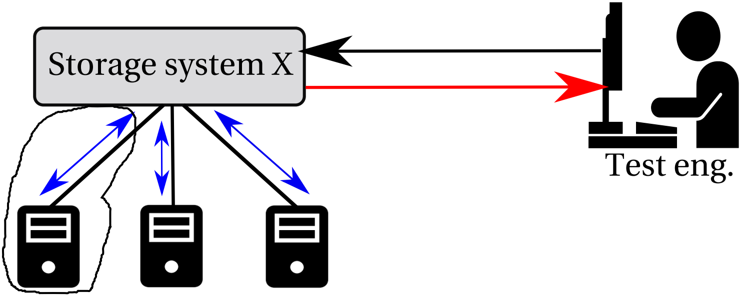
How is I/O performance often evaluated?



⇒ High-level metrics (e.g., overall I/O throughput, exec. time, . . .)

⇒ Results: aggregated and reported by the storage system

How is I/O performance often evaluated?



- $\Rightarrow$ Lack of I/O analysis tools (measurements $\neq$ understanding)
- $\Rightarrow$ Potential I/O errors in lower layers are ignored
- $\Rightarrow$ Nothing is known about workloads' data access
  - $\rightarrow$ Pattern-related errors?

Goal

⇒ Analyzing **I/O patterns**[1] of storage workloads

Requirements

⇒ Flexible and simple as high-level evaluations

⇒ straightforward results

⇒ Work in production environments

→ Negligible overhead

→ Verified behaviours in lower layers

1. We define I/O access pattern of a given workload as the sequences of the I/O requests issued by the target I/O process during a given workload to access on-disk data files

Outline

Context

Tracing in the I/O evaluation context

IOscope[1] design & validation

Experiments on MongoDB & Cassandra

Conclusions

[1] https://github.com/LeUnAiDeS/IOscope *(reproducible scenarios provided)*

Tracing in the I/O evaluation context

⇒ Generic tracing – no filtering mechanism[1]

- → Heavy tracing output
- → Post-analysis effort is not negligible

[1] Betke, E. et al. Real-time i/o-monitoring of hpc applications with siox, elasticsearch, grafana and fuse.High Performance Computing. (2017)

Tracing in the I/O evaluation context

⇒ Generic tracing – no filtering mechanism[1]

- → Heavy tracing output
- → Post-analysis effort is not negligible

⇒ Support partial storage media[2]

- → Works only with HDDs

[2] Daoud, H., Dagenais, M.R. : Recovering disk storage metrics from low-level trace events. Software : Practice and Experience 48(5), (2018)

Tracing in the I/O evaluation context

- ⇒ Generic tracing – no filtering mechanism[1]
  - → Heavy tracing output
  - → Post-analysis effort is not negligible
- ⇒ Support partial storage media[2]
  - → Works only with HDDs
- ⇒ Partial coverage of filesystems[3]

Tracing in the I/O evaluation context

$\Rightarrow$ Generic tracing – no filtering mechanism[1]

- $\rightarrow$ Heavy tracing output
- $\rightarrow$ Post-analysis effort is not negligible

$\Rightarrow$ Support partial storage media[2]

- $\rightarrow$ Works only with HDDs

$\Rightarrow$ Partial coverage of filesystems[3]

[3] Jeong, S. et al. : Androstep : Android storage performance analysis tool. In : Software Engineering (Workshops). vol. 13, pp. 327–340 (2013)

Tracing in the I/O evaluation context

⇒ Generic tracing – no filtering mechanism[1]
  - → Heavy tracing output
  - → Post-analysis effort is not negligible

⇒ Support partial storage media[2]
  - → Works only with HDDs

⇒ Partial coverage of filesystems[3]

⇒ Modified kernel is required[3]

[3] Jeong, S. et al. : Androstep : Android storage performance analysis tool. In : Software Engineering (Workshops). vol. 13, pp. 327–340 (2013)

Tracing in the I/O evaluation context

⇒ Generic tracing – no filtering mechanism[1]

→ Heavy tracing output

→ Post-analysis effort is not negligible

⇒ Support partial storage media[2]

→ Works only with HDDs

⇒ Partial coverage of filesystems[3]

⇒ Modified kernel is required[3]

⇒ *Many tracing frameworks for I/O extrapolation*[4,5,6,7,8]

→ Different scope – traces are more important than workloads!

[4] Luo, X et al. : Hpc i/o trace extrapolation. In : Proceedings of the 4th Workshop on Extreme Scale Programming Tools. (2015)
[5] Luo, X et al. : Scalaioextrap : Elastic i/o tracing and extrapolation. In : Parallel and Distributed Processing Symposium (IPDPS), (2017)
[6] Chahal, D et al. : Performance extrapolation of io intensive workloads : Work in progress. 7th International Conf. on Perf. Eng. (2016)
[7] Virk, R et al. : Trace replay based i/o performance studies for enterprise workload migration. In : 2nd Annual Conference of CMG India. (2015)
[8] Tak, B et al. : Pseudoapp : Performance prediction for application migration to cloud. In : Integrated Network Management (IM 2013)

Tracing in the I/O evaluation context

- ⇒ Generic tracing – no filtering mechanism[1]
  - → Heavy tracing output
  - → Post-analysis effort is not negligible
- ⇒ Support partial storage media[2]
  - → Works only with HDDs
- ⇒ Partial coverage of filesystems[3]
- ⇒ Modified kernel is required[3]
- ⇒ *Many tracing frameworks for I/O extrapolation*[4,5,6,7,8]
  - → Different scope – traces are more important than workloads!
- ⇒ Tools like *DTrace, SystemTap, LTTng* use dynamic loading
  - → Usage suitability is affected (e.g., signed kernels, no compilation tools)

Tracing in the I/O evaluation context

⇒ Generic tracing – no filtering mechanism[1]
  → Heavy tracing output
  → Post-analysis effort is not negligible

⇒ Support partial storage media[2]
  → Works only with HDDs

⇒ Partial coverage of filesystems[3]

⇒ Modified kernel is required[3]

⇒ *Many tracing frameworks for I/O extrapolation*[4,5,6,7,8]
  → Different scope – traces are more important than workloads!

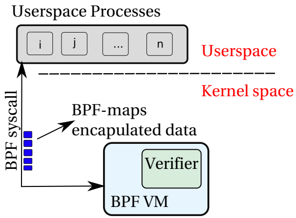
⇒ Tools like *DTrace, SystemTap, LTTng* use dynamic loading
  → Usage suitability is affected (e.g., signed kernels, no compilation tools)

⇒ Partial coverage of I/O methods such as mmap[9,10,11]

[9] Mantri, S.G. : Efficient In-Depth IO Tracing and its application for optimizing systems. Virginia Tech (2014)
[10] IOVISOR BCC Project's slower tools (fileslower, ext4slower).
[11] IOVISOR BCC Project's IOsnoop tools.

IOscope is based on eBPF. What is eBPF?

- ⇒ A recent tracing and filtering technology
- ⇒ Connect to all data sources: (Kprobes, Uprobes, tracepoints, ...)
- ⇒ Almost near-zero overhead (4 ns per *syscall*) [2]
- ⇒ Formally adopted by the Linux kernel ($\geq$ Linux 3.19)
- ⇒ Has a lot of front-end projects like *IOVISOR's BCC*
  - → No more byte code!
  - → Towards precise-objective tracing



eBPF's basic components

2. Starovoitov, A. : https ://lwn.net/Articles/598545/ (2014)

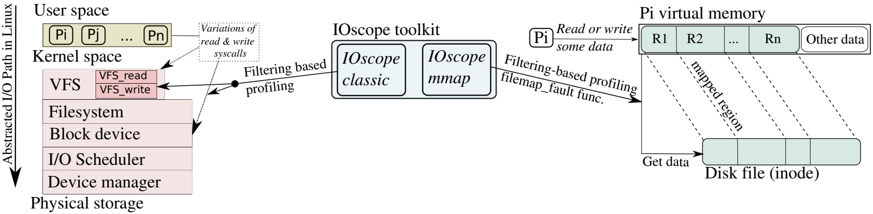
IOscope tracer: uncovering I/O patterns for storage workloads

⇒ *filtering-based tracing* mechanism [3]
- → Reduce the collected data by an order of magnitude
- → Less interceptions = lower overhead (*leq* 0.08%)
- → Tiny tracing granularities (e.g. R/W operations)

⇒ Two tools for:
- → I/O workloads issued using *syscalls*
- → memory mapped-files workloads

⇒ Collect specific data {*file offsets, size, latency, timestamps, op.Mode*}

⇒ Useful for in-production usage



IOscope overall design

3. Several filters are applied in both kernel and userspace pair during collecting traces (pid, files, I/O operations, . . .)

IOscope validation

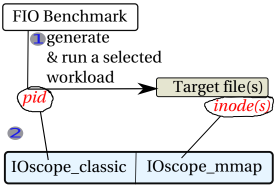
⇒ Linux kernel 4.9.0

⇒ Flexible Input/Output (FIO) benchmark is used

→ Generate diverse workloads / several I/O methods

E.g. rand reading workload for mmap :

```
fio -name=testfile -rw=randread
-ioengine=mmap -direct=0 -size=10G
-numjobs=1 -group_reporting
```



How IOscope catches I/O traces

TABLE – *Validated I/O access modes and workloads*

Fio IOengine	Target syscalls	Tested workloads : *read, write, randread, randwrite, readwrite*, and *randreadwrite*
Sync	read, write	all
Psync	pread, pwrite	all
Pvsync	preadv, pwritev	all
Pvsync2	preadv2, pwritev2	all
posixaio	aio_read, aio_write	all
Mmap	mmap, memcpy	all

IOscope validation

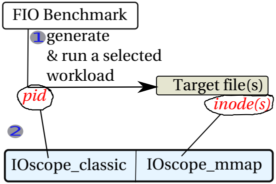
⇒ Linux kernel 4.9.0

⇒ Flexible Input/Output (FIO) benchmark is used

→ Generate diverse workloads / several I/O methods

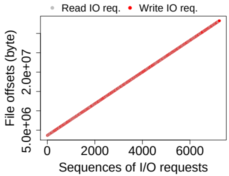
E.g. rand reading workload for mmap :

```
fio -name=testfile -rw=randread
-ioengine=mmap -direct=0 -size=10G
-numjobs=1 -group_reporting
```

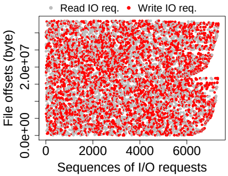


How IOscope catches I/O traces

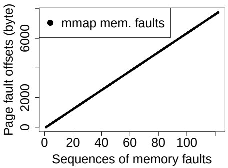
Some results of validated workloads:



rw workload - Poxisaio IOengine



randrw workload - Psync IOengine



r workload - mmap IOengine

Objective: Uncovering potential pattern-related issues

Experimental setup

⇒ MongoDB & Cassandra are tested
  → MongoDB v3.4 with WiredTiger (classic I/O)
  → Cassandra v3.0.14 (mmap I/O)

⇒ One client to index a simple *int* field

⇒ Single server & two-shards cluster configurations

⇒ Hash sharding for clustered configuration – load balancing

⇒ Experiments run on HDDs & SSDs separately

⇒ Cache is cleaned between experiments

⇒ Data contiguity is tested using *FIBMAP*

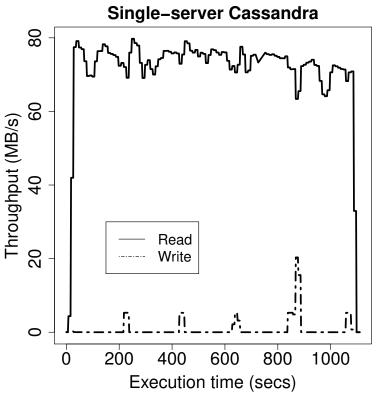
Datasets

⇒ Two equally-sized datasets (same characteristics)

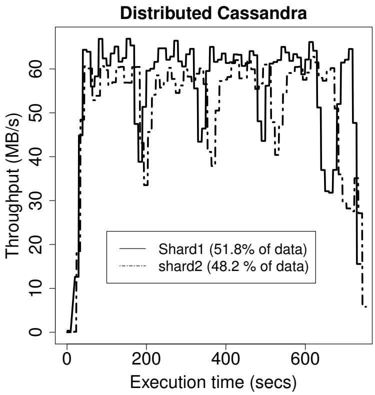
(min, avrg, max) in KB	N. of data units	Size (Gb)
(1, 3.47, 6)	20,000,000 docs	71

⇒ Elements x (int, date, 2 x str[min, max], array[1..4] x string[min, max])

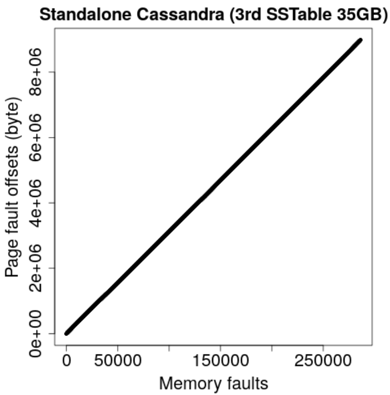
Experimentation - Cassandra results



I/O throughput of a single server experiment(HDD)

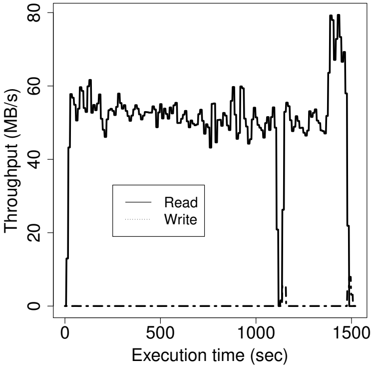


I/O throughput of Cassandra's two-nodes cluster (HDD)

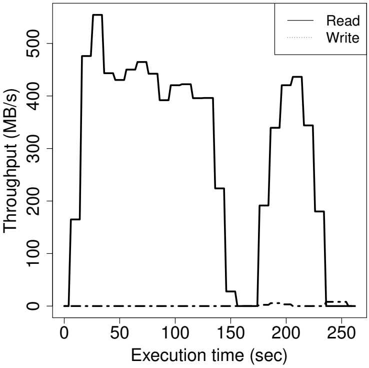


I/O pattern of single-server experiment (HDD)

→ Clustered experiments also have pure sequential access pattern



Single-server experiment – HDD



Single-server experiment – SSD

Experimentation - MongoDB results

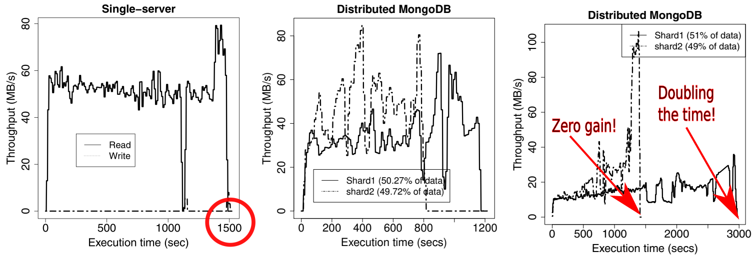


FIGURE – Results on HDD. Single-server & two different runs of distributed experiments

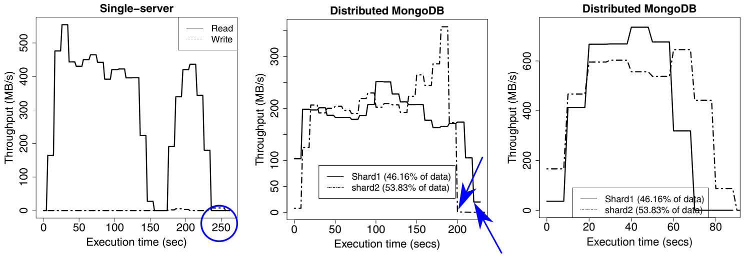
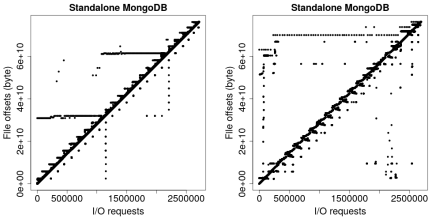
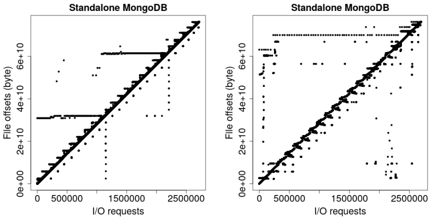


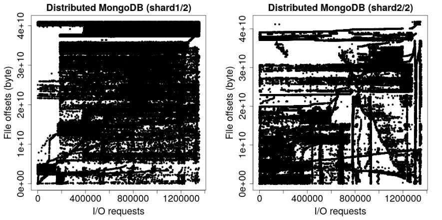
FIGURE – Results on SSD. Single-server & two different runs of distributed experiments



Single-server I/O patterns: HDD (left) and SSD (right)

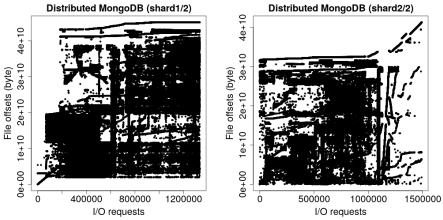


Single-server I/O patterns: HDD (left) and SSD (right)



HDD – second clustered experiment

- ⇒ Max overhead is less than 0.80%
- ⇒ Acceptable access pattern on single-server experiments
- ⇒ Seq. access → Random access on shards!
- ⇒ Data distribution issue
- ⇒ SSDs are affected by I/O patterns too!



SSD – first clustered experiment

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

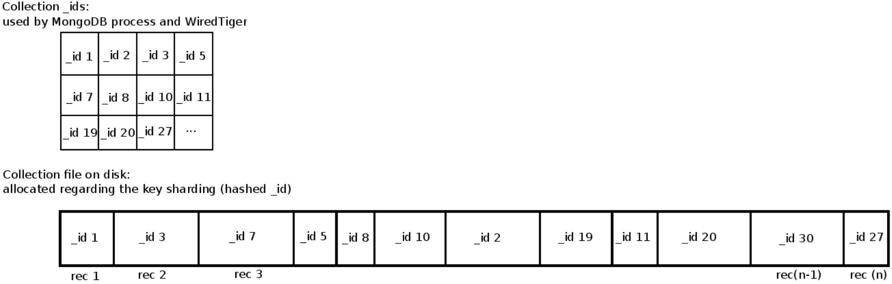


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

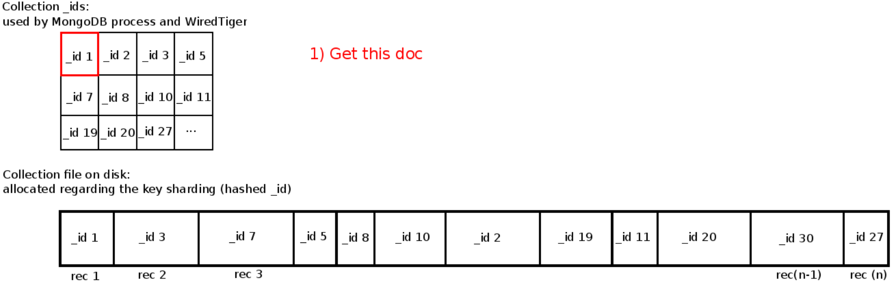


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

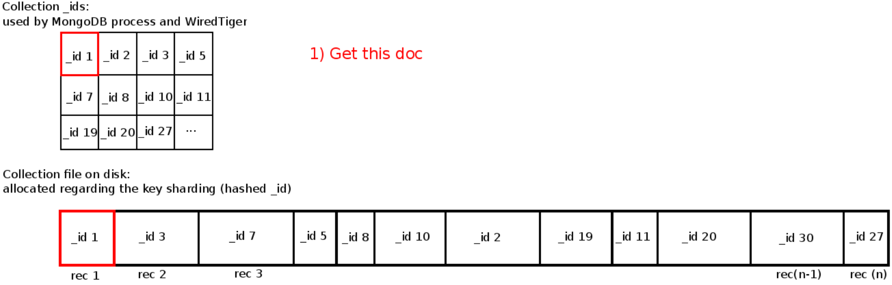


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

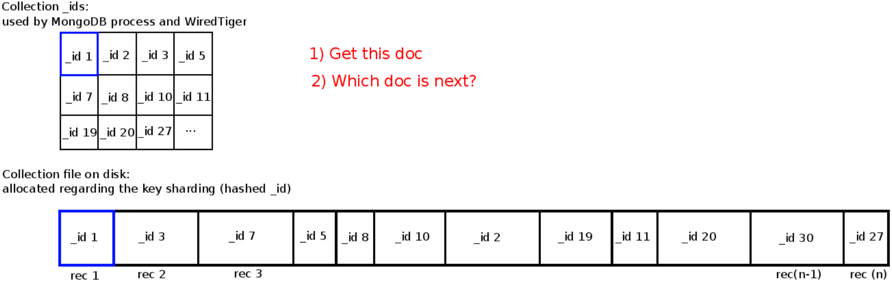


FIGURE – MongoDB scanning table Vs records' order on the disk

Experimentation - MongoDB results

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

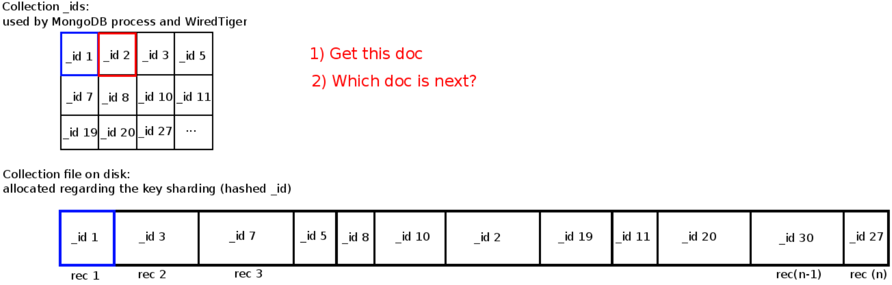


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

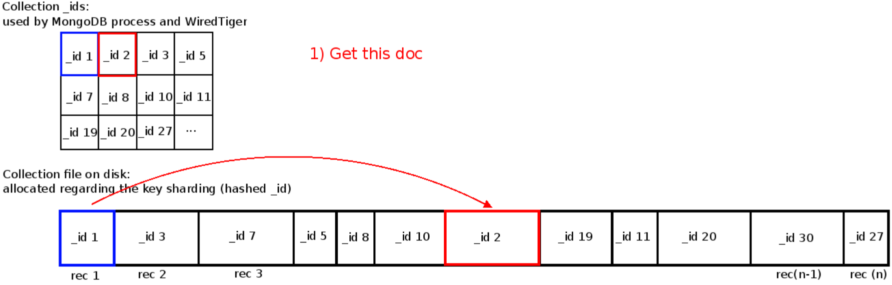


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

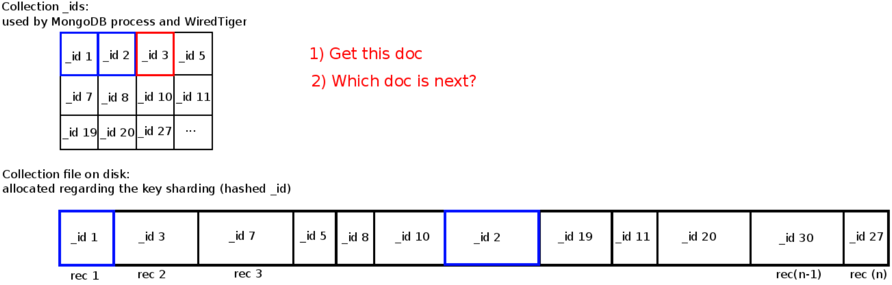


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

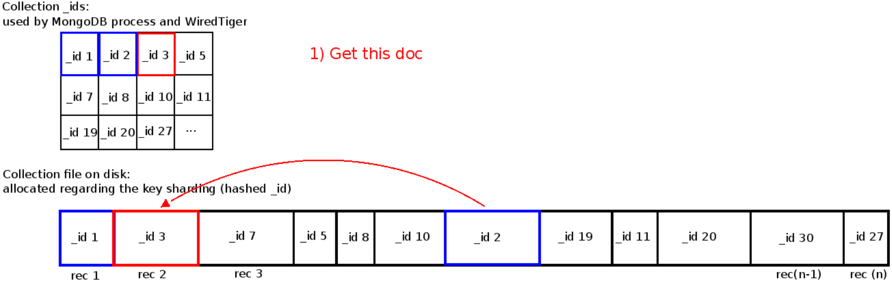


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

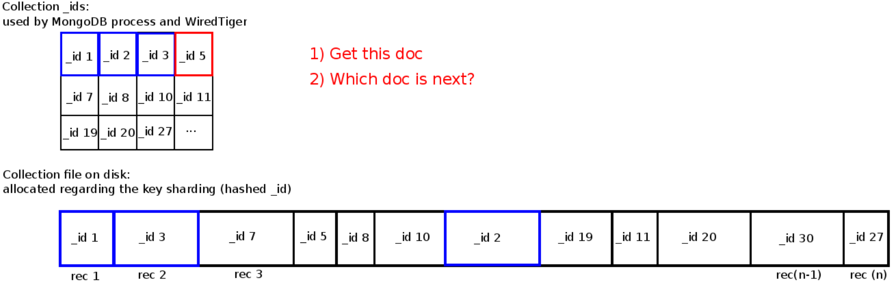


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

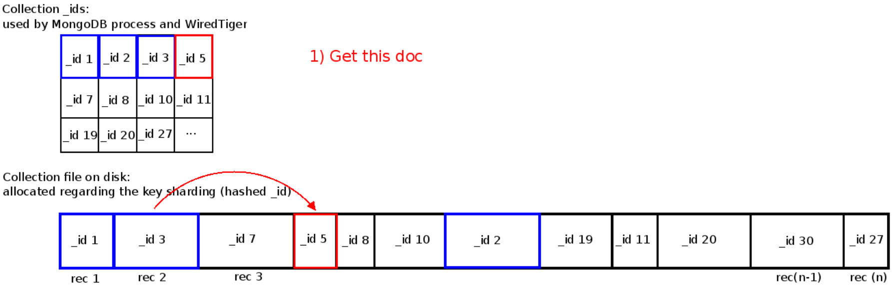


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

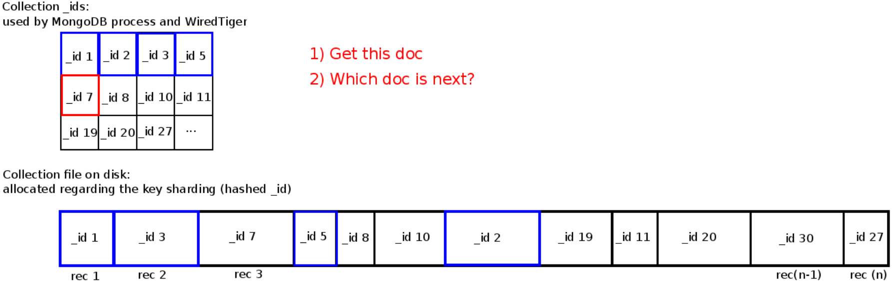


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

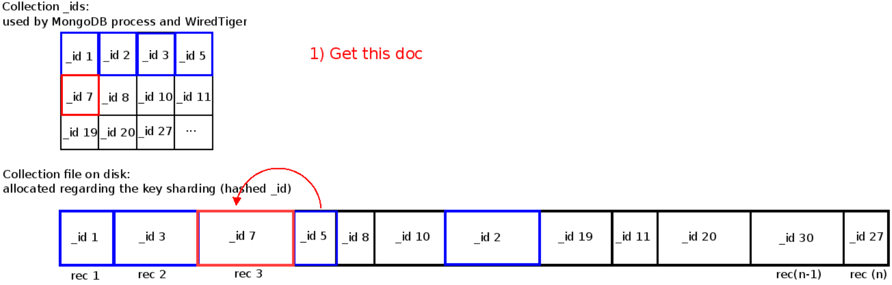


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

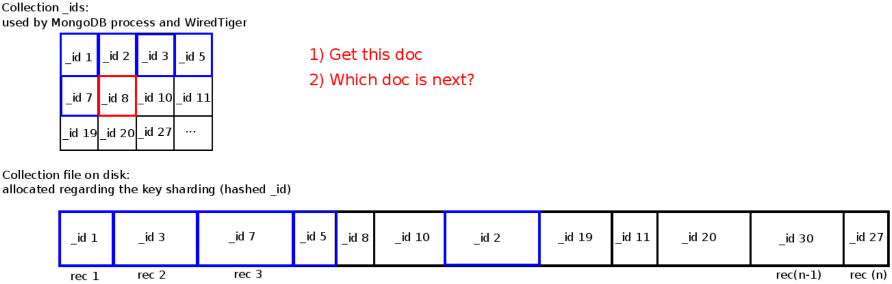


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

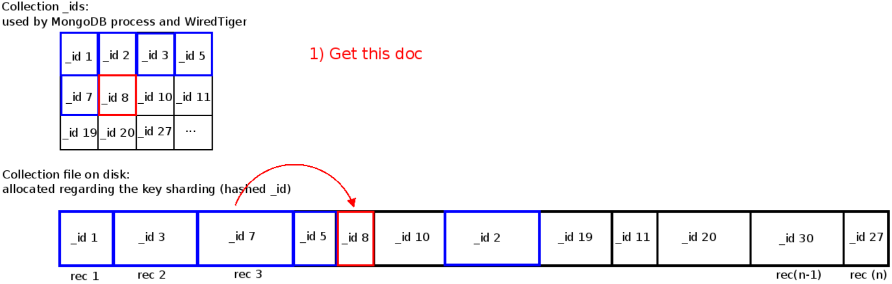


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

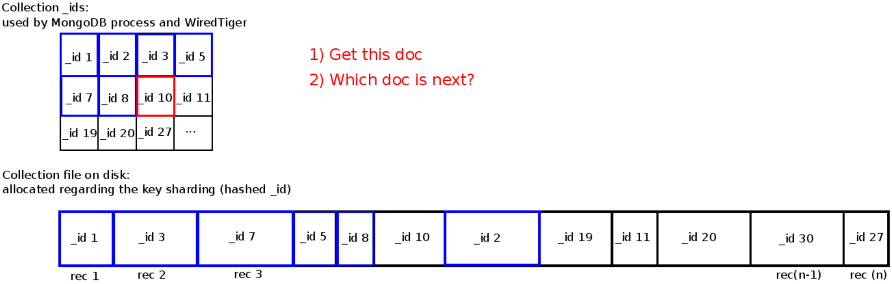


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

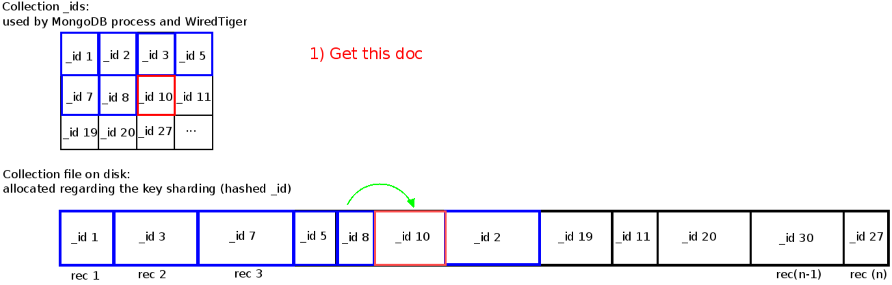


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

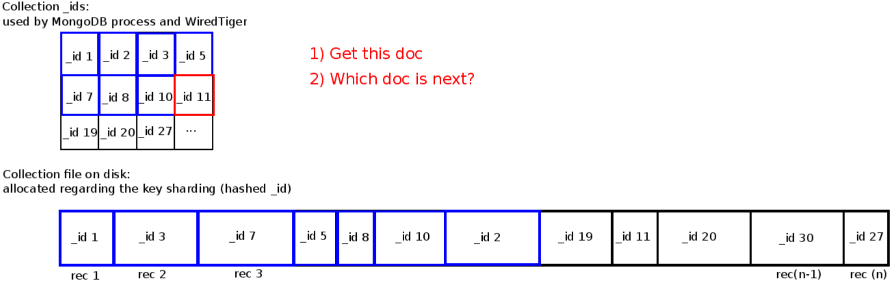


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

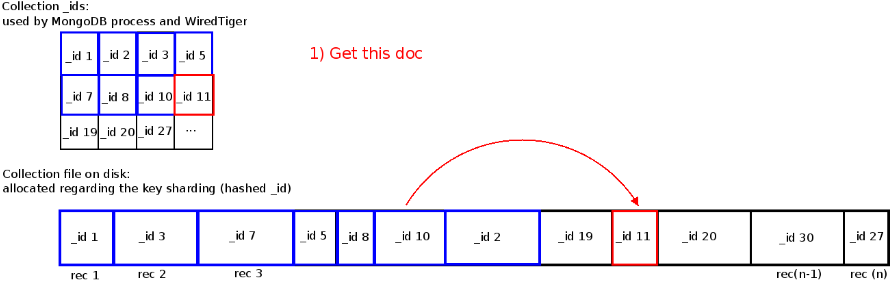


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

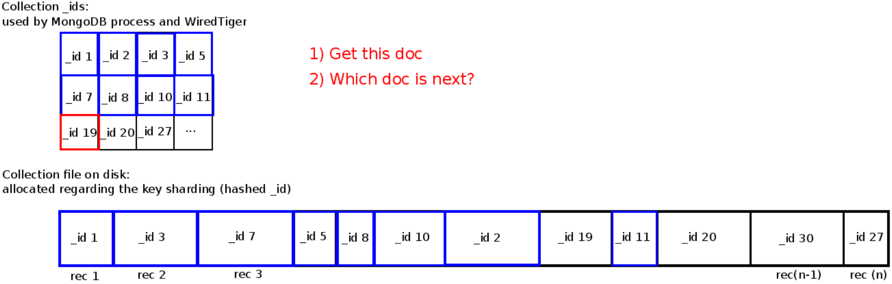


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

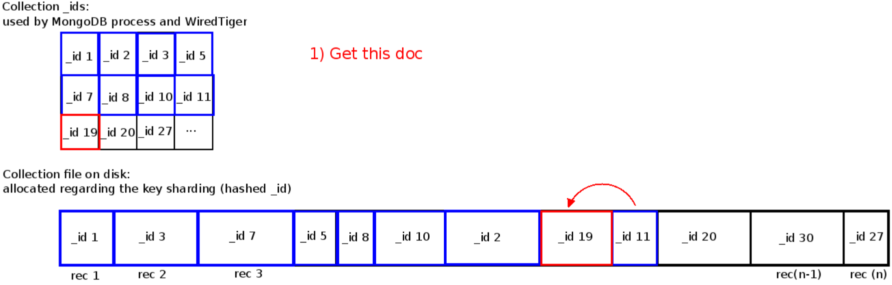


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

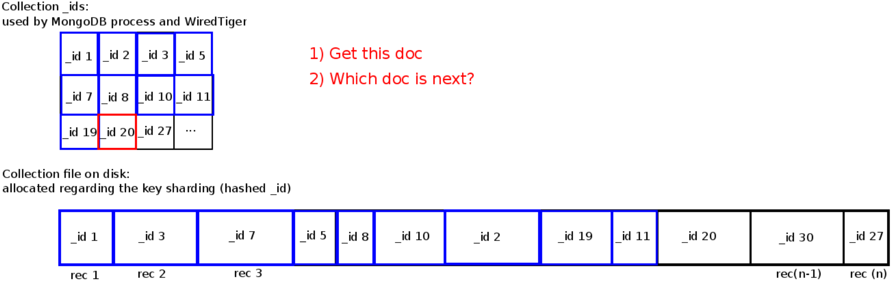


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

$\Rightarrow$ Mismatch between the scanning table vs data stored on disk

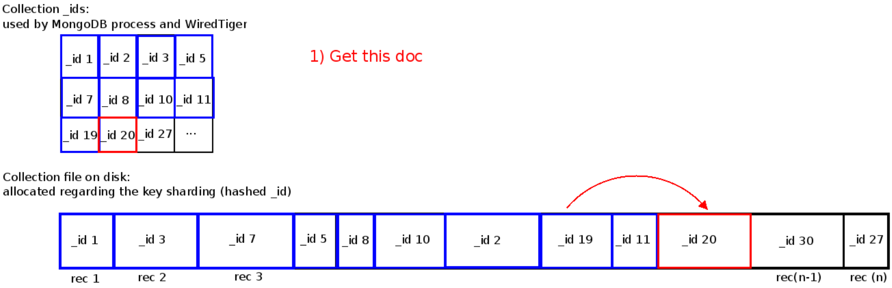


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

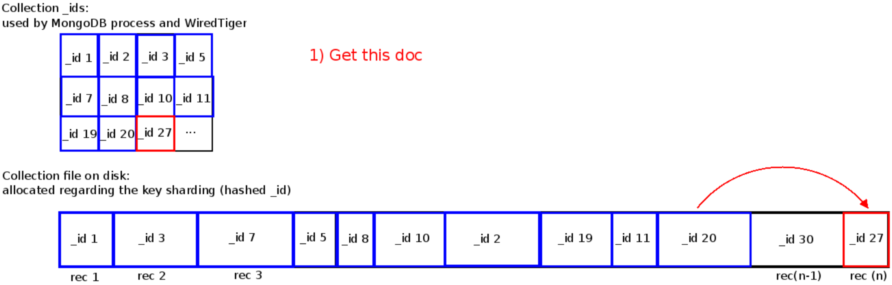


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

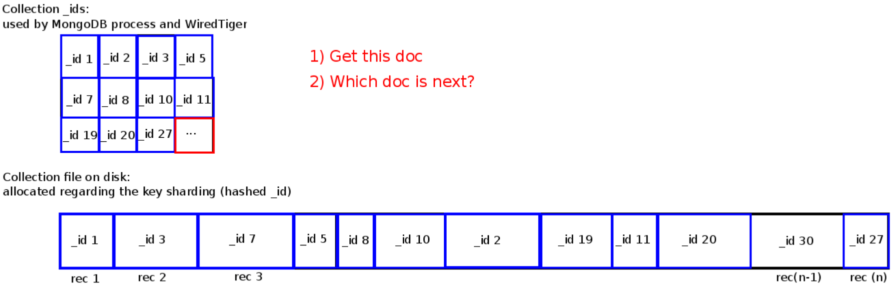


FIGURE – MongoDB scanning table Vs records' order on the disk

The exact issue

⇒ Mismatch between the scanning table vs data stored on disk

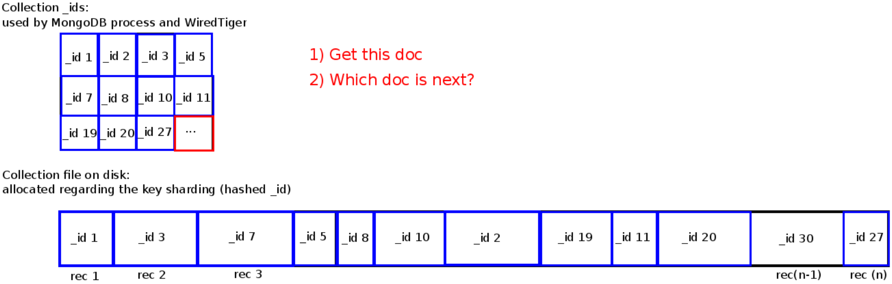


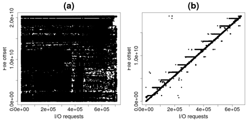
FIGURE – MongoDB scanning table Vs records' order on the disk

We proposed an ad-hoc solution

⇒ Key-idea: rewrite shards data

⇒ MongoDB updates its view of data

⇒ High cost, but gives insights!



I/O access pattern a) before & b) after applying

Conclusions

- ⇒ Low-level I/O evaluations are not negligable to preserve I/O performance
- ⇒ Systems' complexity may hide issues
  - → We showed how an unexpected issue affects the performance of *MongoDB*
- ⇒ IOscope is proposed to analyse I/O patterns of storage systems
- ⇒ We demonstrated how it is worthy to use IOscope to go beyond high-level evaluations' results

Future work

- ⇒ Extend IOscope to uncover other I/O-related issues
- ⇒ Performing more performance evaluations on other storage systems
- ⇒ Further investigation on SSDs and I/O patterns

Conclusions

- ⇒ Low-level I/O evaluations are not negligable to preserve I/O performance
- ⇒ Systems' complexity may hide issues
  - → We showed how an unexpected issue affects the performance of *MongoDB*
- ⇒ IOscope is proposed to analyse I/O patterns of storage systems
- ⇒ We demonstrated how it is worthy to use IOscope to go beyond high-level evaluations' results

Future work

- ⇒ Extend IOscope to uncover other I/O-related issues
- ⇒ Performing more performance evaluations on other storage systems
- ⇒ Further investigation on SSDs and I/O patterns

Questions are welcome !