Performance Testing nopayloaddb

Lino Gerlach, Ruslan Mashinistov, Paul Laycock

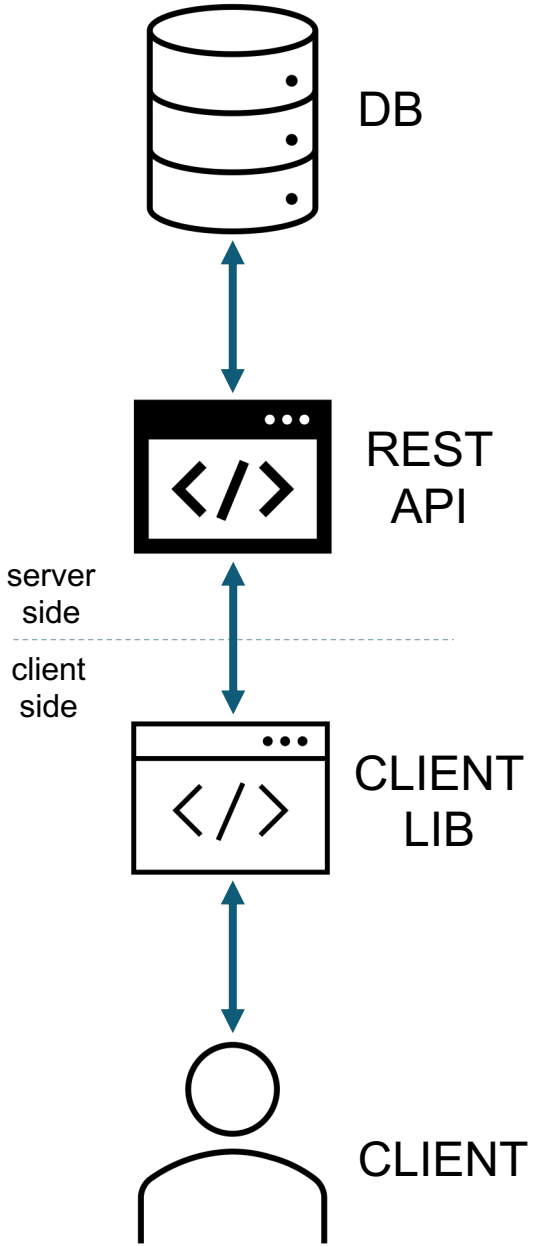
02.05.2023

@BrookhavenLab

Overview

Testing nopayloaddb performance

- Access patterns & DB scenarios
- Performance metrics
- HTC vs Concurrent requests
  - Pre-emptive vs Cooperative Concurrency



Testing nopayloaddb - Strategy

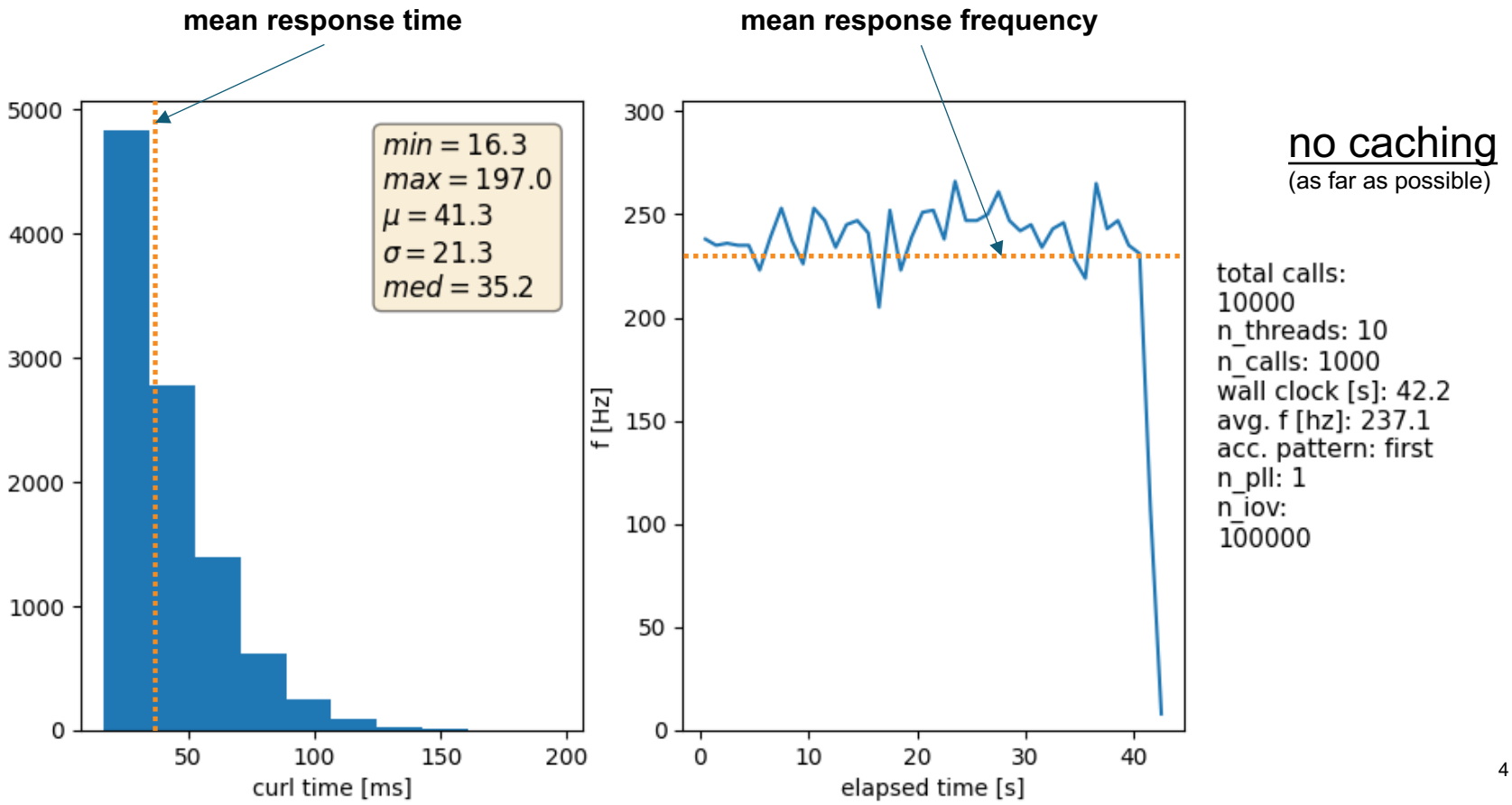
Ansatz: Simulate expected DB occupancy & access patterns

- DB occupancy: Scenarios w/ varying number of types & IOVs
- Access patterns: Conditions DB accessed in offline reco (HTC)
  - Make parallel requests using HTCondor or Multithreading
    - Request random or specific constant IOVs
  - Response times and -frequencies as performance metrics

Scenario	Global Tags	Payload Types	Payload IOVs (per type)	Update Rate
tiny	1	10	100 (10)	
tiny-moderate	1	10	2000 (200)	
moderate	1	100	20000 (200)	1 day
heavy-usage	1	100	500000 (5000)	1 hour
worst-case	1	200	5200000 (26000)	10 minutes

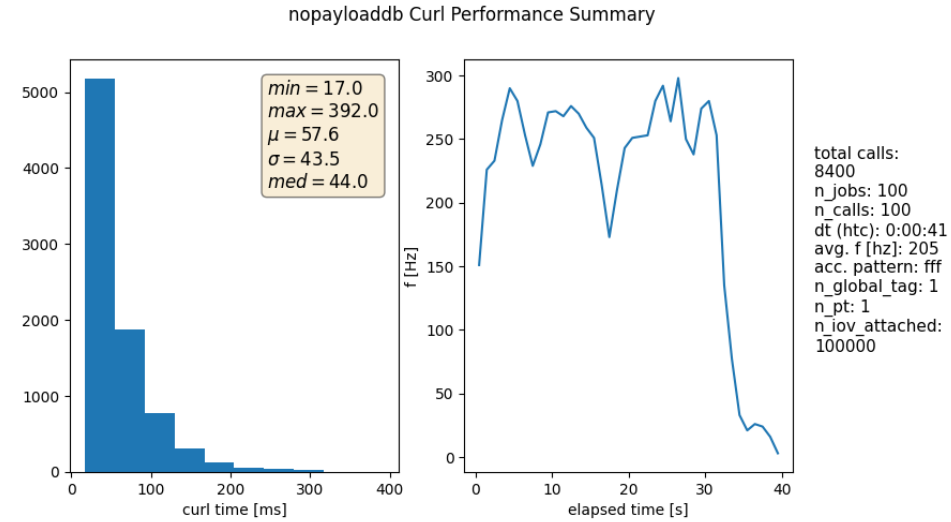
Test Campaign - Example

- Example results of a test campaign (absolute values don't matter here)
- Scaling tests: single number as metric of whole campaign

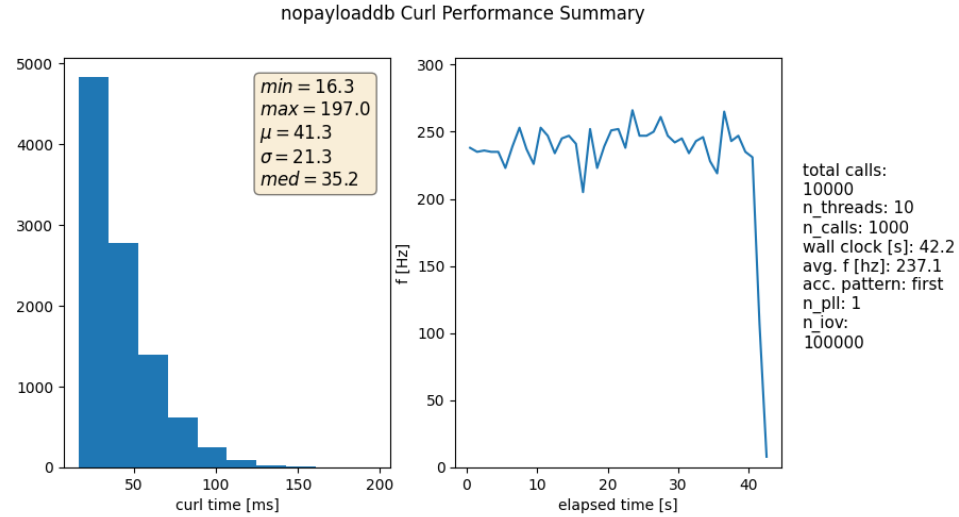


HTCondor vs Concurrent tests

HTC (good daily form)



Concurrent tests



HTCondor tests:

- frequency fluctuates w/ number of running jobs
  - Reaches peak w/ delay
  - Tail: last job(s) still running

Concurrent tests:

- frequency more constant
  - Reaches peak immediately
  - Drops of sharply towards end

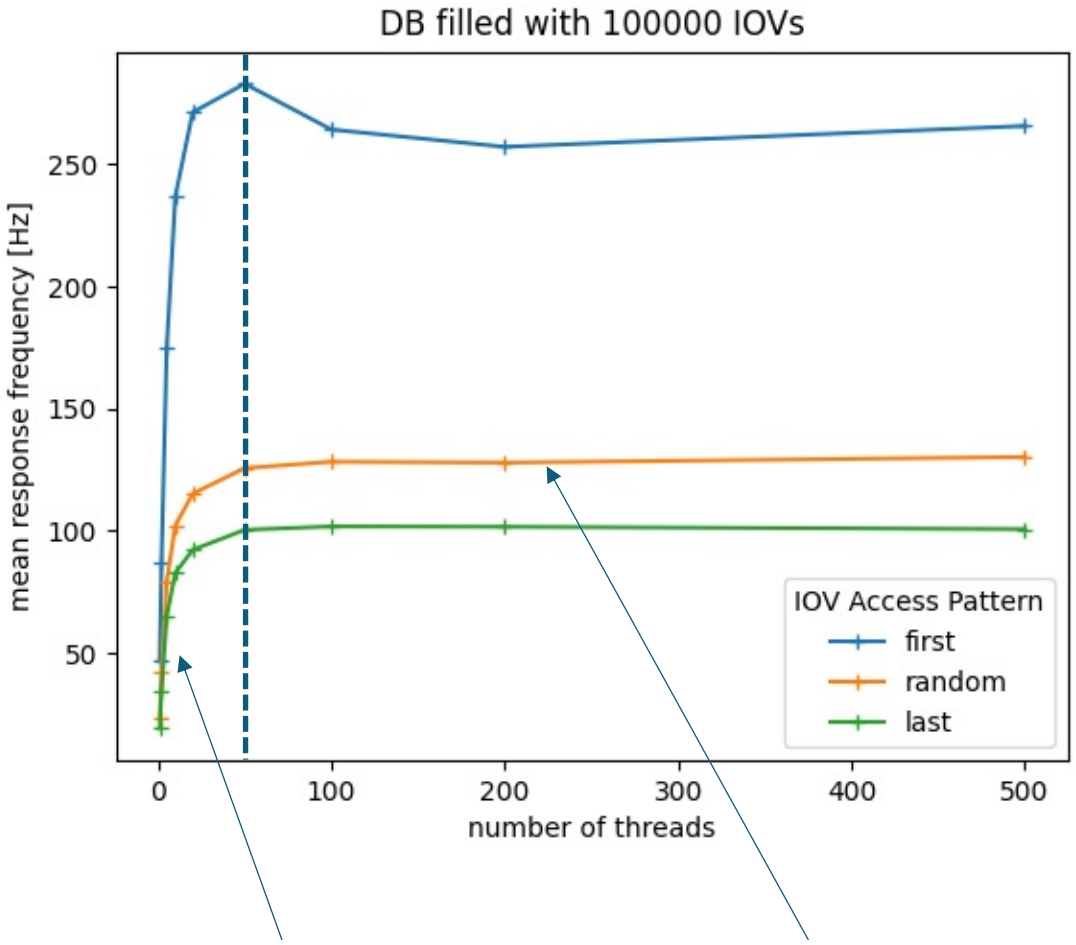
Avg. frequencies very similar

(Keep in mind tail in HTC case)

- Determined by server-side

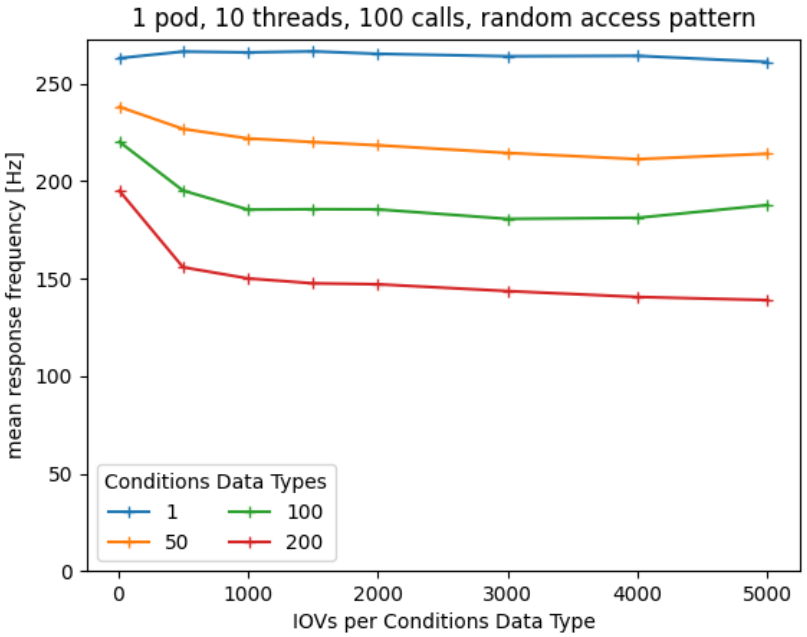
Multithreaded Tests – Number of Threads

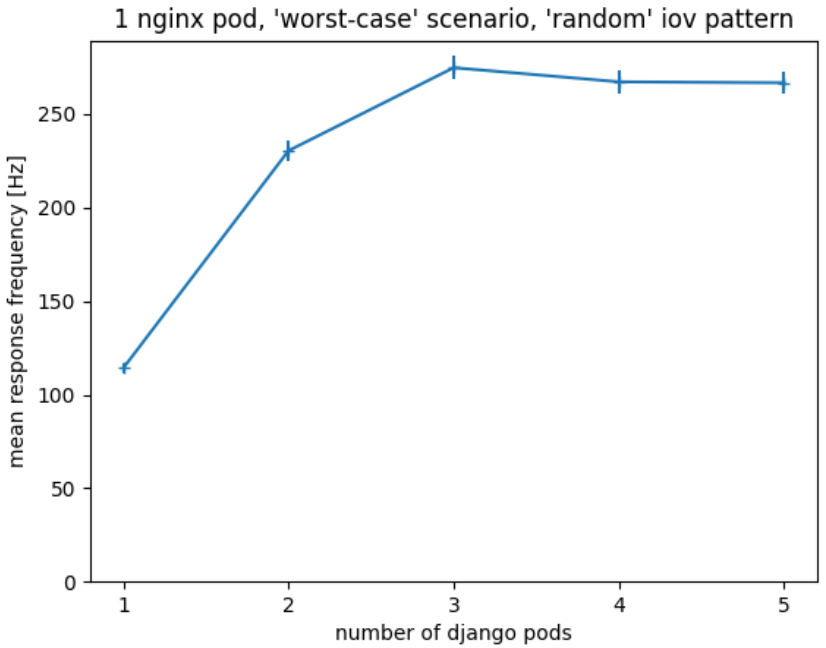
Each data point represents test campaign with 10k total calls



- Multithreaded requests can reach server-side bottleneck

Performance Testing – Results

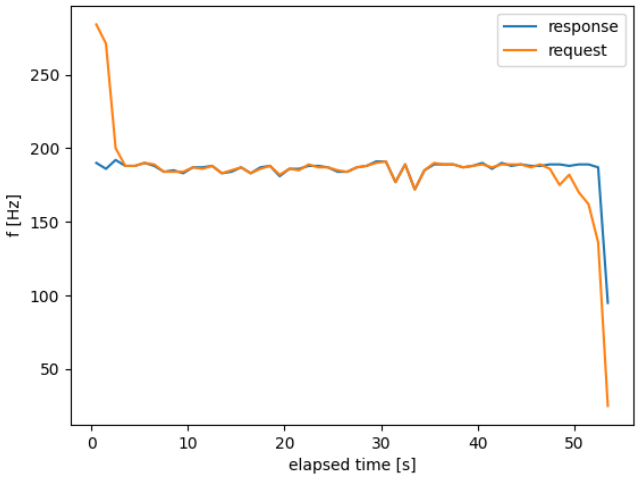




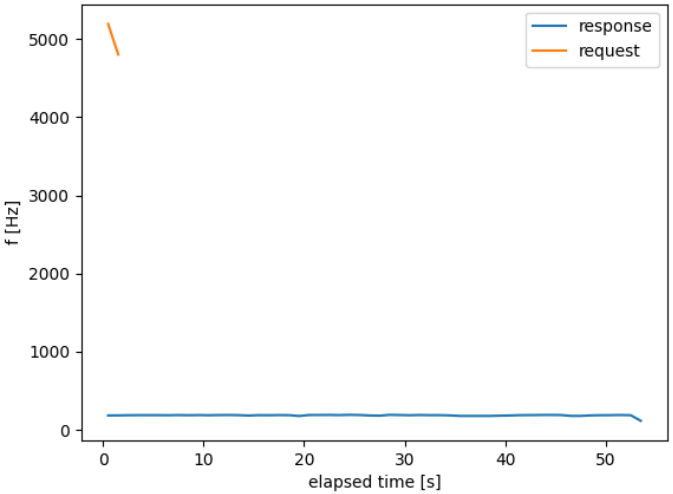
- Multithreaded tests allow nice scaling plots like these:
  - Mean response frequency vs number of IOVs & Payload Types
  - Mean response frequency vs number of Django pods (deployment)

Simulating Peak Access Rates

threading (200)



asyncio



- Scenario where O(100k) Jobs are launched
  - **threading** does not reach high peak access frequencies
  - **asyncio** does at first glance, but nginx logs tell otherwise
    - Slow log writing? / Only written when request is being processed?

```
/var/log/nginx
$ tail -10000 access.log | awk 'NR % 1000 == 0'
10.130.2.1 - - [28/Apr/2023:13:20:50 +0000] "GET /api/cdb_rest/payloa
02101 HTTP/1.1" 200 12347 "-" "Python/3.10 aiohttp/3.8.3"
10.130.2.1 - - [28/Apr/2023:13:20:53 +0000] "GET /api/cdb_rest/payloa
671 HTTP/1.1" 200 12385 "-" "Python/3.10 aiohttp/3.8.3"
10.130.2.1 - - [28/Apr/2023:13:20:55 +0000] "GET /api/cdb_rest/payloa
965 HTTP/1.1" 200 12169 "-" "Python/3.10 aiohttp/3.8.3"
10.130.2.1 - - [28/Apr/2023:13:20:58 +0000] "GET /api/cdb_rest/payloa
075 HTTP/1.1" 200 11961 "-" "Python/3.10 aiohttp/3.8.3"
10.130.2.1 - - [28/Apr/2023:13:21:01 +0000] "GET /api/cdb_rest/payloa
89422 HTTP/1.1" 200 12387 "-" "Python/3.10 aiohttp/3.8.3"
10.130.2.1 - - [28/Apr/2023:13:21:04 +0000] "GET /api/cdb_rest/payloa
3097 HTTP/1.1" 200 12381 "-" "Python/3.10 aiohttp/3.8.3"
10.130.2.1 - - [28/Apr/2023:13:21:06 +0000] "GET /api/cdb_rest/payloa
0414 HTTP/1.1" 200 12177 "-" "Python/3.10 aiohttp/3.8.3"
10.130.2.1 - - [28/Apr/2023:13:21:09 +0000] "GET /api/cdb_rest/payloa
1708 HTTP/1.1" 200 12373 "-" "Python/3.10 aiohttp/3.8.3"
10.130.2.1 - - [28/Apr/2023:13:21:12 +0000] "GET /api/cdb_rest/payloa
7376 HTTP/1.1" 200 12373 "-" "Python/3.10 aiohttp/3.8.3"
10.130.2.1 - - [28/Apr/2023:13:21:14 +0000] "GET /api/cdb_rest/payloa
4075 HTTP/1.1" 200 12177 "-" "Python/3.10 aiohttp/3.8.3"
$
```

Conclusion & Outlook

Test **nopayloaddb** Performance simulating DB occupancy & access patterns

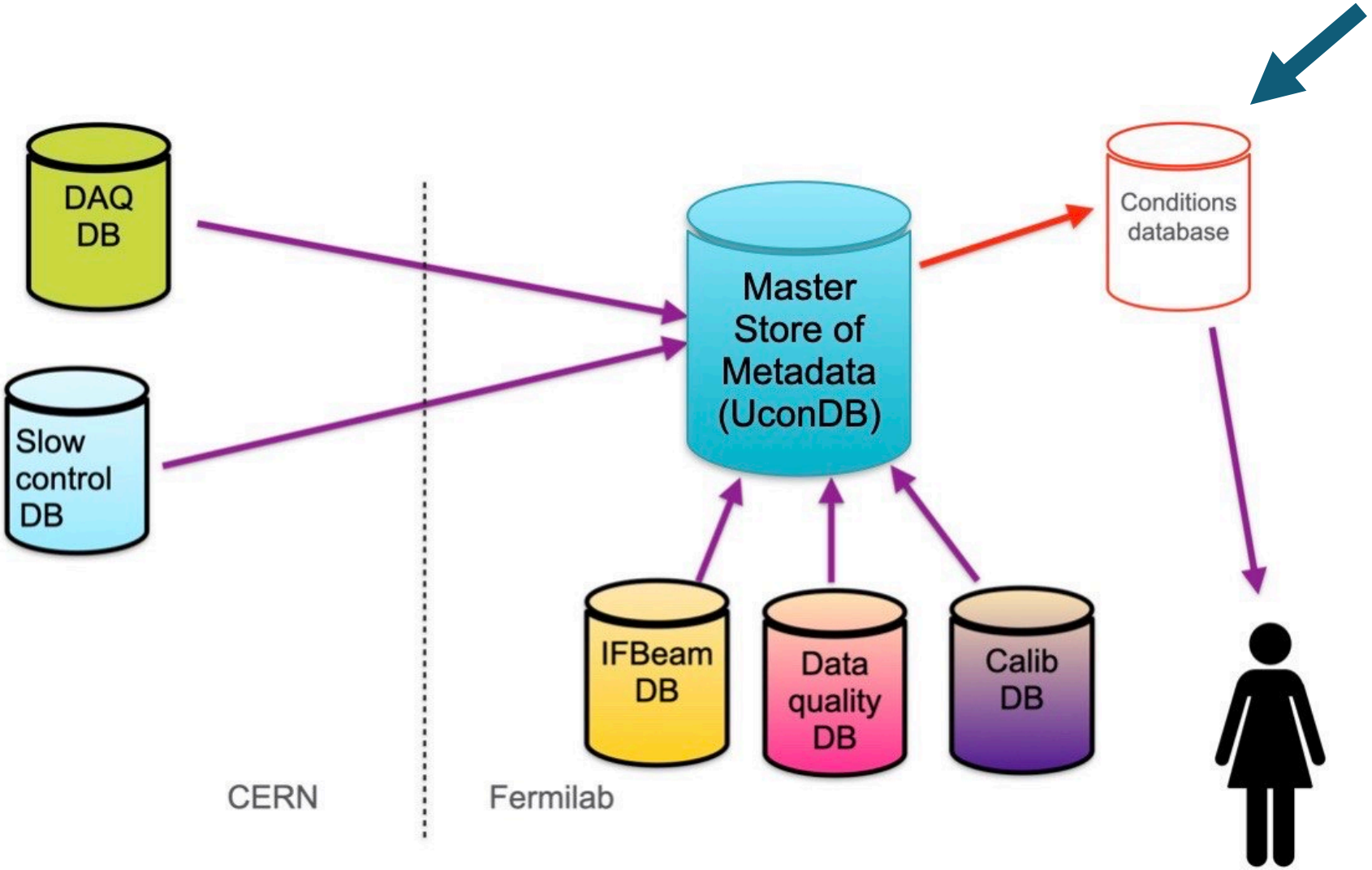
- HTC-driven test results vary widely, long turn-around time
- Concurrent requests from single machine reach server-side bottleneck reliably
- Understand mean response frequency & mean response time scaling w.r.t. DB occupancy and deployment config
- Try to simulate peak request rate scenario
  - Cooperative concurrent requests (**asyncio)** seem promising
    - Still some open questions
- Might consider dedicated tools (**wrk, locust, …)**
- Any thoughts / ideas / recommendations?

Brookhaven National Laboratory

U.S. DEPARTMENT OF ENERGY

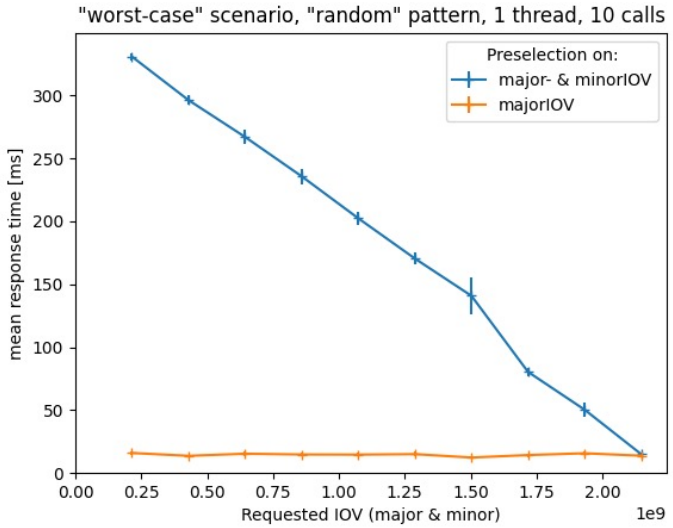
Backup Slides

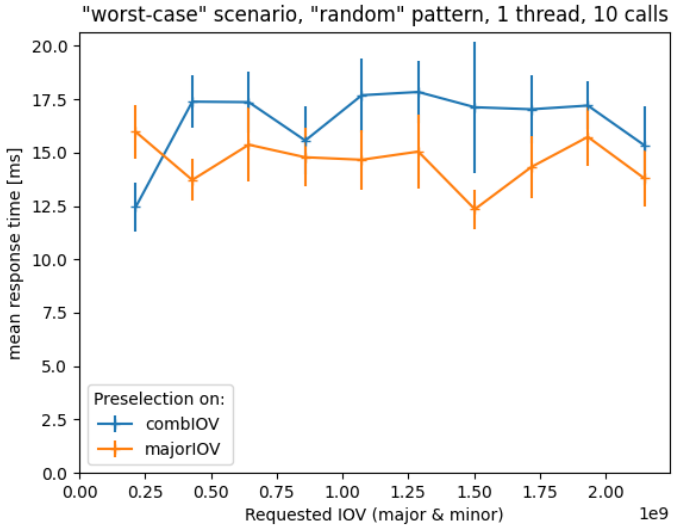
Overview – ProtoDUNE Databases



from Ana Paula Vizcaya Hernandez

Raw SQL - Combined IOV Column





- Preselection on major- & minor IOV (AND / OR)
  - Scales with entries to consider
  - Query uses ‘Filter’
- Preselection on single column (<=)
  - Constant time
  - Query uses ‘Index Condition’

- Combine major- and minor IOV into single column:

major_iov	minor_iov	comb_iov
477658914	1001747433	477658914.0000000001001747433
23283443	1525747152	23283443.0000000001525747152
1834979804	648013294	1834979804.0000000000648013294
bigint	bigint	decimal(38, 19)

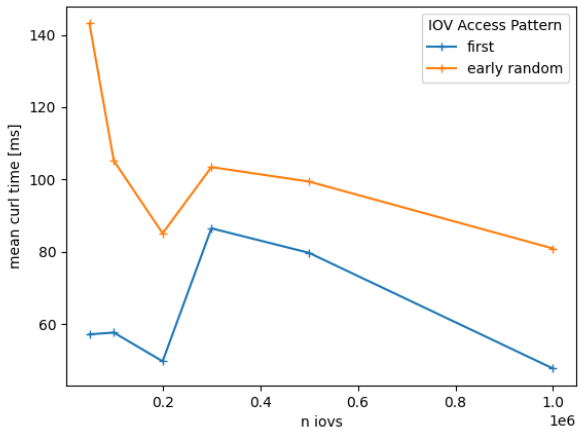
- Fast across all values while selecting on both

HTCondor Performance Tests - Problems

- Jobs are not always starting at the same time
  - Cannot precisely control total request frequency
- Some nodes are slow
  - Blacklisting not feasible (often discover new ones)
  - On-the-fly node performance test -> neglect job if failed
    - Total number of requests varies
- Networking performance varies over time
  - Normalize response times by ‘google.com’ response time
- Writing jobs’ output affected by file system performance
  - Use /gpfs/ (hard to say how large impact is)

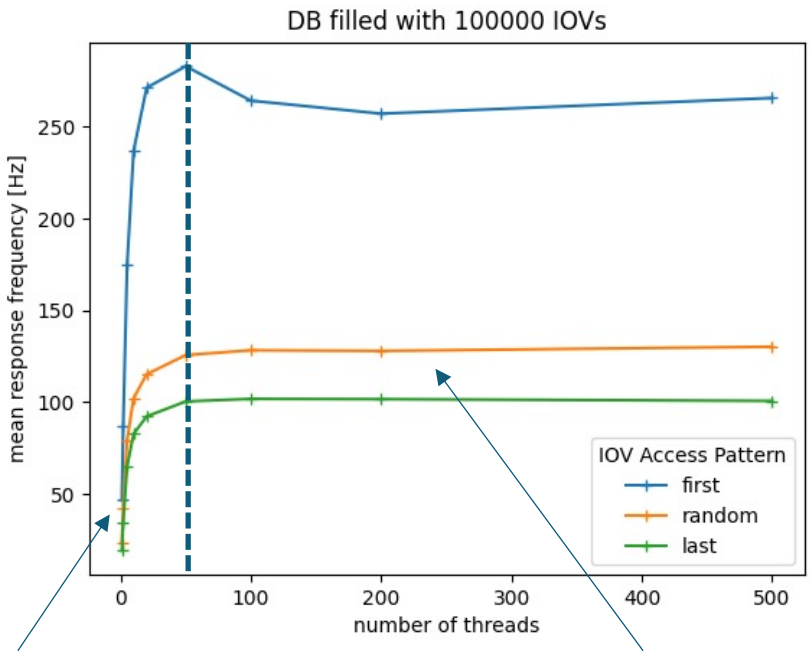
Identical tests yield strongly varying results over time.

- Difficult to precisely measure scaling
- Long turn-around for simple checks



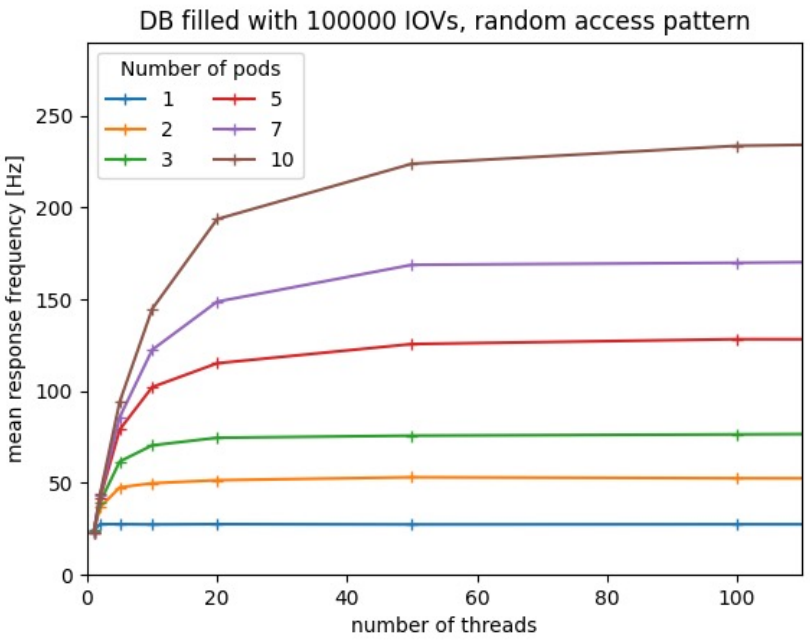
Multithreaded Tests – Threads & Pods

Each data point represents test campaign with 10k total calls



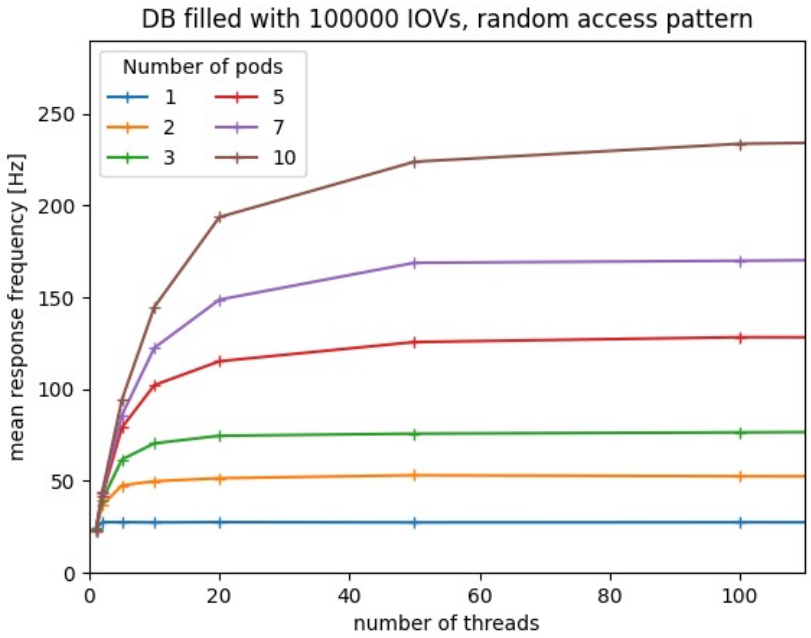
limited by client-side request frequency

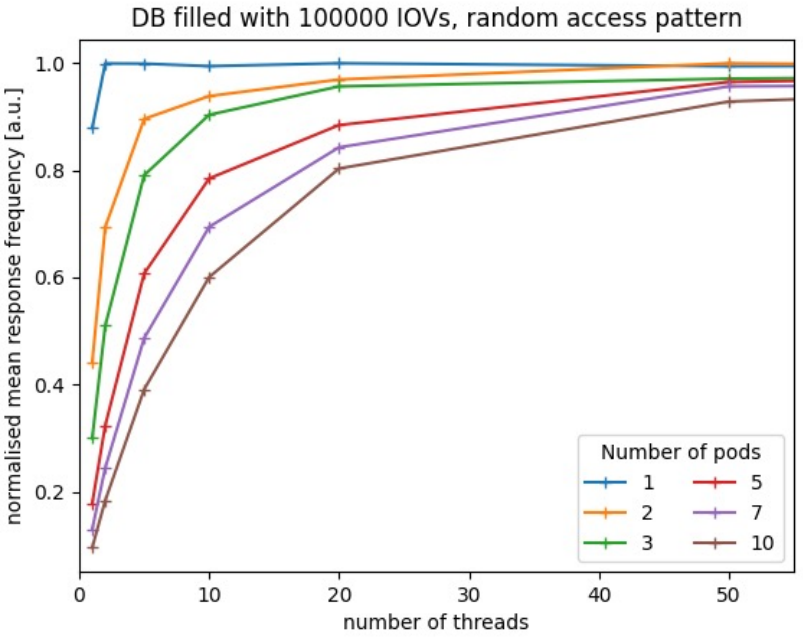
limited by server-side response frequency



- Multithreaded requests can reach server-side bottleneck
  - Number of required threads scales with number of pods

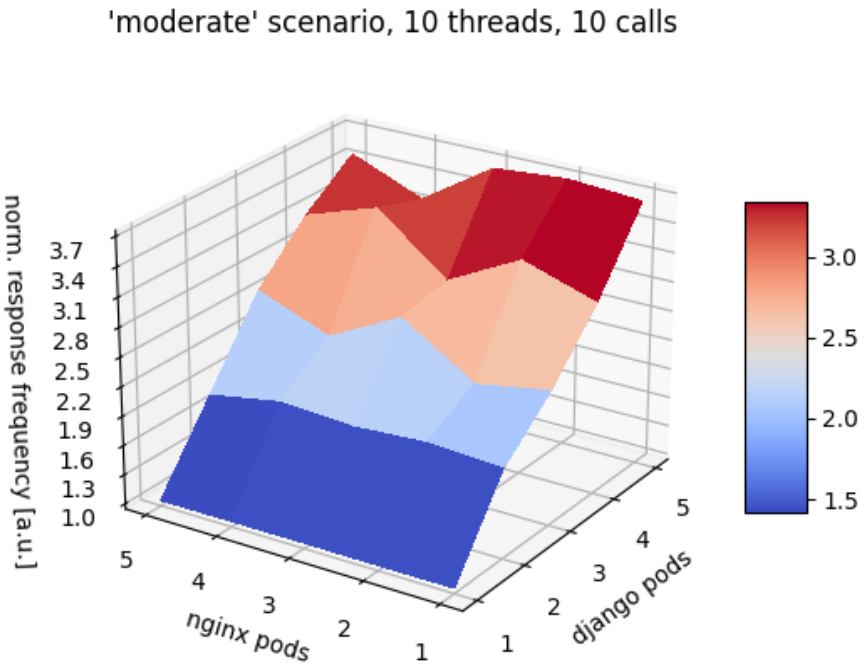
MT Testing – Scaling w/ Pods

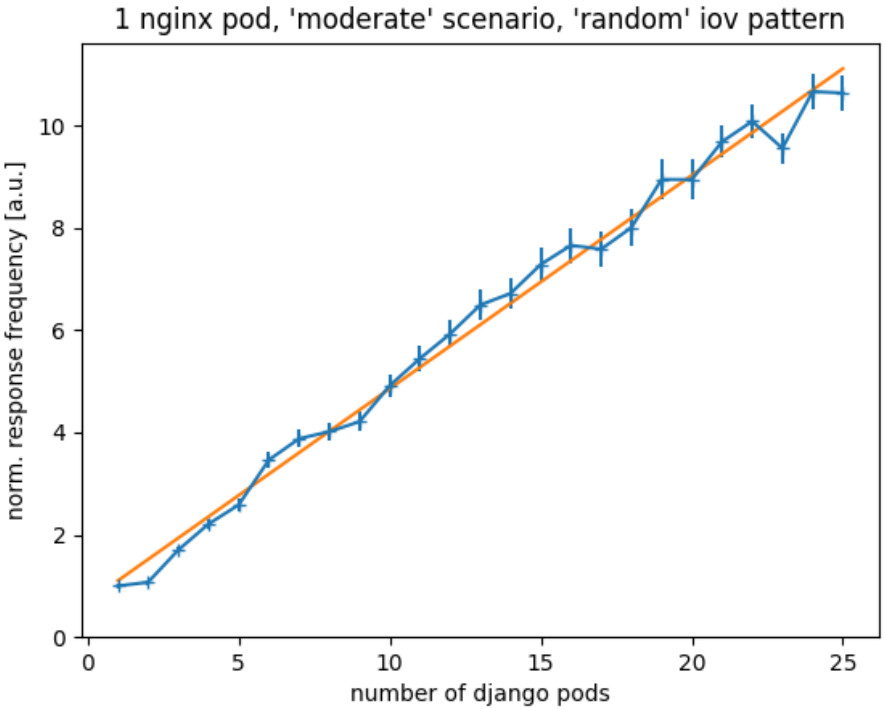




- Number of threads to reach max response frequency
  - Scale w/ number of pods
  - 2 threads for 1 pod, ~100 threads for 10 pods

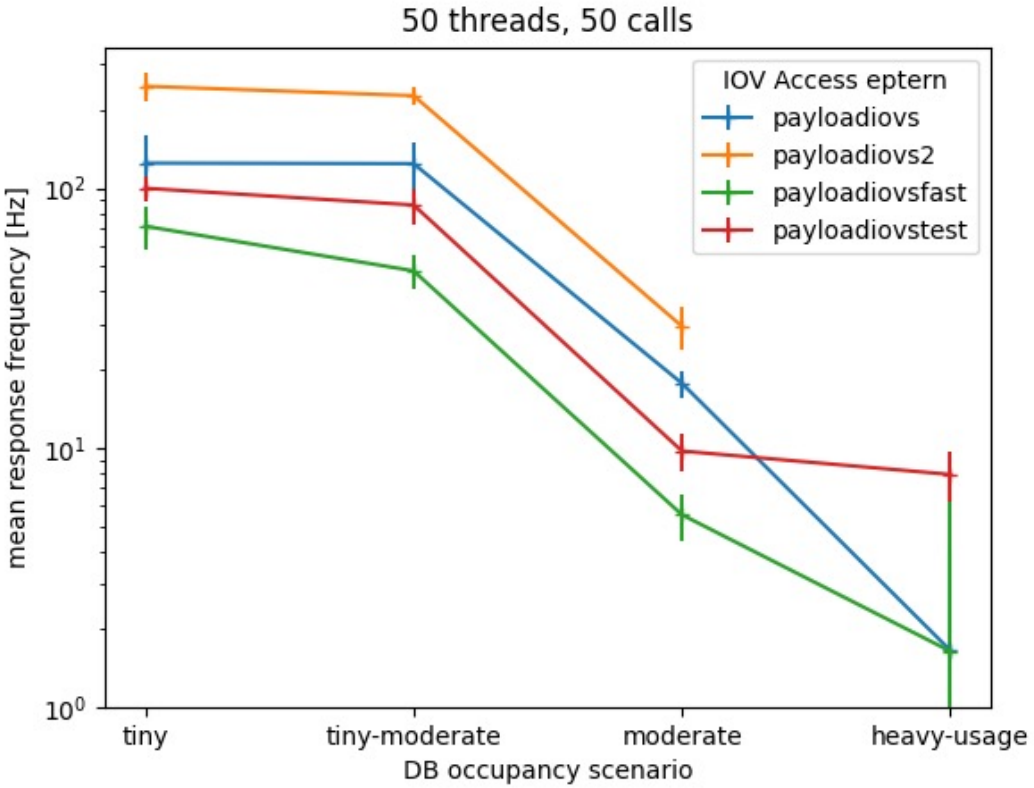
Scaling w/ Pods (Django vs Nginx)





- Mean response frequency depends linearly on Django pods
- Number of nginx pods has no significant impact

Scaling with DB Scenarios - ORM



Different implementations using ORM

What every query has to solve:

- Filter on global tag
- Filter on major- and minorIOV *
- Find 'latest' IOV for each payloadtype **

*: my_major<majorIOV OR
(my_major=majorIOV AND my_minor<=minorIOV)

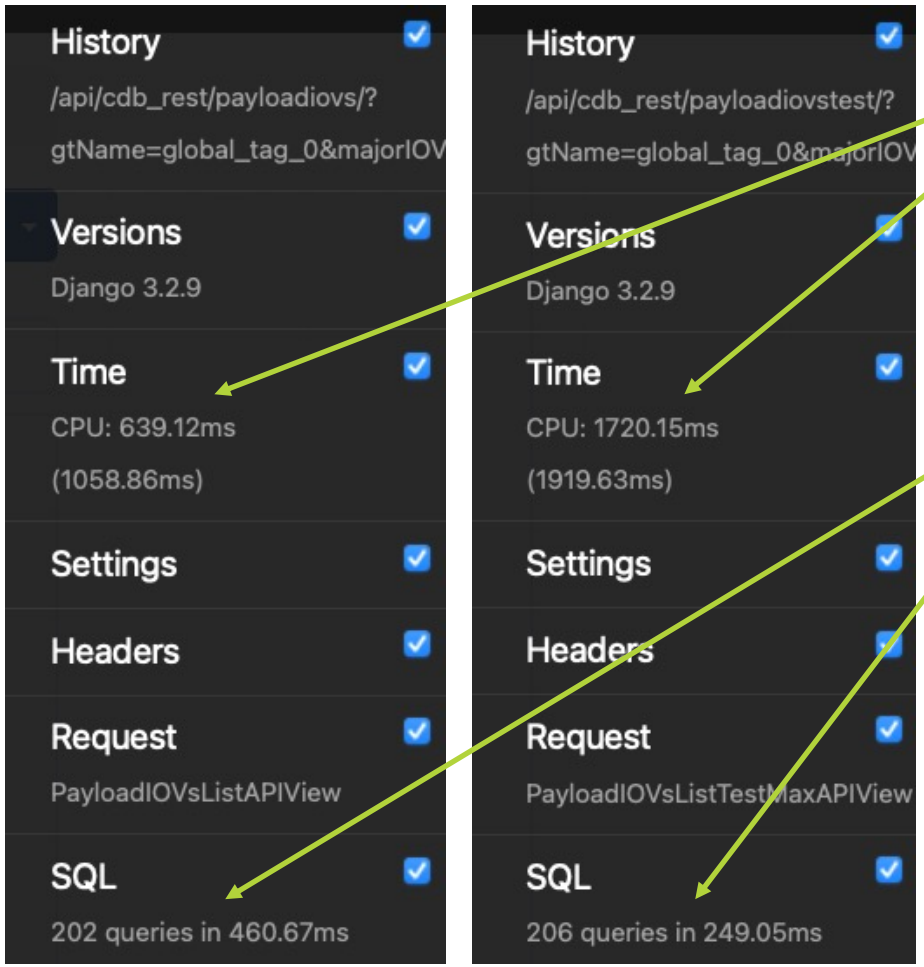
**: for max majorIOV, find max minorIOV

Even fastest endpoint drops below 10 Hz for 'heavy-usage'

- Too slow, need to improve!

SQL Query Times – Django Toolbar

- Deploy ‘django-debug-toolbar’ (GUI in browser)



Full request duration (misleading because of DEBUG mode)

Waiting for DB response

Computationally most expensive query:
- Find latest iov for each payload list
- Test raw SQL queries for this subproblem

Raw SQL Queries (Simplified)

- Consider only major iov for simplicity
  - Time different raw SQL queries by hand

'heavy-usage' scenario

Name	SQL String	Time [ms]	Comments

distinct	`SELECT DISTINCT ON (payload_list_id)`
	`  payload_list_id, payload_url, major_iov`
	`FROM`
	`  "PayloadIOV"`
	`ORDER BY`
	`  payload_list_id, major_iov DESC;`
		300	Most straight forward
			Rather slow

group (self-join)	`SELECT m.payload_list_id, m.payload_url, t.mx`
	`FROM (`
	`  SELECT payload_list_id, MAX(major_iov) AS mx`
	`  FROM "PayloadIOV"`
	`  GROUP BY payload_list_id`
	`) t JOIN "PayloadIOV" m ON m.payload_list_id = t.payload_list_id`
	`AND t.mx = m.major_iov;`
		65	Postgres-specific hack using self-join to return any column when using GROUP BY

lateral	`SELECT u.id, l.major_iov, l.payload_url`
	`FROM  "PayloadList" u`
	`CROSS  JOIN LATERAL (`
	`  SELECT l.major_iov, l.payload_url`
	`  FROM  "PayloadIOV" l`
	`  WHERE  l.payload_list_id = u.id`
	`  ORDER  BY l.major_iov DESC NULLS LAST`
	`  LIMIT  1`
	`  ) l;`
		5	Superfast, needs additional index
			
			`CREATE INDEX covering_idx`
			`ON "PayloadIOV"`
			`(payload_list_id, major_iov`
			`DESC NULLS LAST)`
			`INCLUDE (payload_url);`

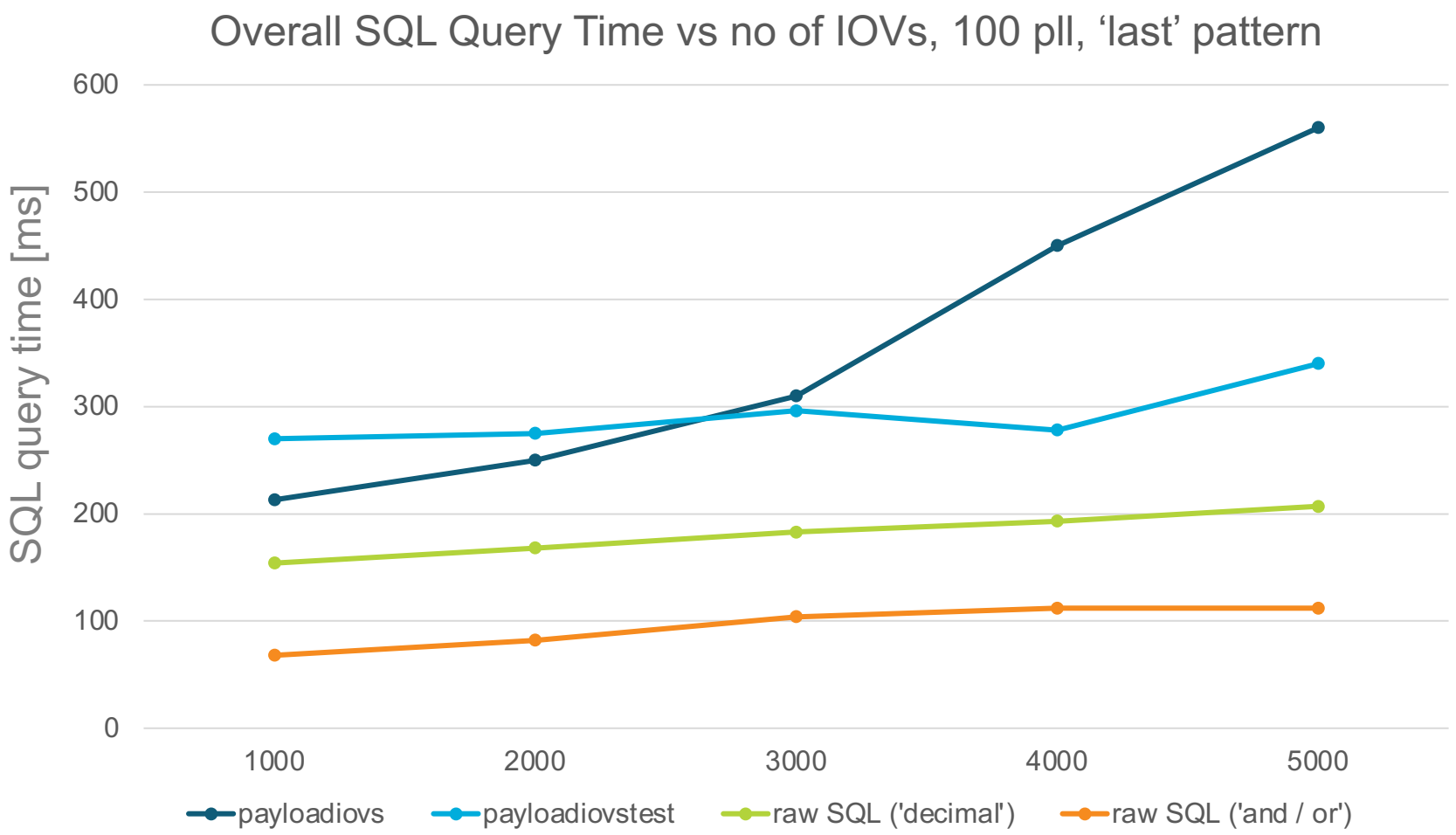
Full Raw SQL Query ‘group’

```
WITH major_max_table AS(
    SELECT m.payload_list_id, m.payload_url, m.major_iov, m.minor_iov, m.major_iov_end, m.minor_iov_end
    FROM (
        SELECT payload_list_id, MAX(major_iov) AS major_max
        FROM "PayloadIOV"
        WHERE ((major_iov < :my_major_iov) OR (major_iov = :my_major_iov AND minor_iov <= :my_minor_iov))
        AND payload_list_id IN (
            SELECT id FROM "PayloadList"
            WHERE global_tag_id = (
                SELECT id FROM "GlobalTag"
                WHERE name=:'my_gt'
            )
        )
    GROUP BY payload_list_id
    ) t JOIN "PayloadIOV" m ON m.payload_list_id = t.payload_list_id AND t.major_max = m.major_iov
),

major_minor_max_table AS(
    SELECT n.payload_list_id, n.payload_url, n.major_iov, n.minor_iov, n.major_iov_end, n.minor_iov_end
    FROM (
        SELECT payload_list_id, MAX(minor_iov) AS minor_max
        FROM major_max_table
        GROUP BY payload_list_id
    ) u JOIN major_max_table n ON n.payload_list_id = u.payload_list_id AND u.minor_max = n.minor_iov
)

SELECT y.name AS payload_type_name, x.payload_url, x.major_iov, x.minor_iov, x.major_iov_end, x.minor_iov_end
FROM
  major_minor_max_table x
  JOIN "PayloadList" z ON x.payload_list_id = z.id
  JOIN "PayloadType" y ON z.payload_type_id = y.id
;
```

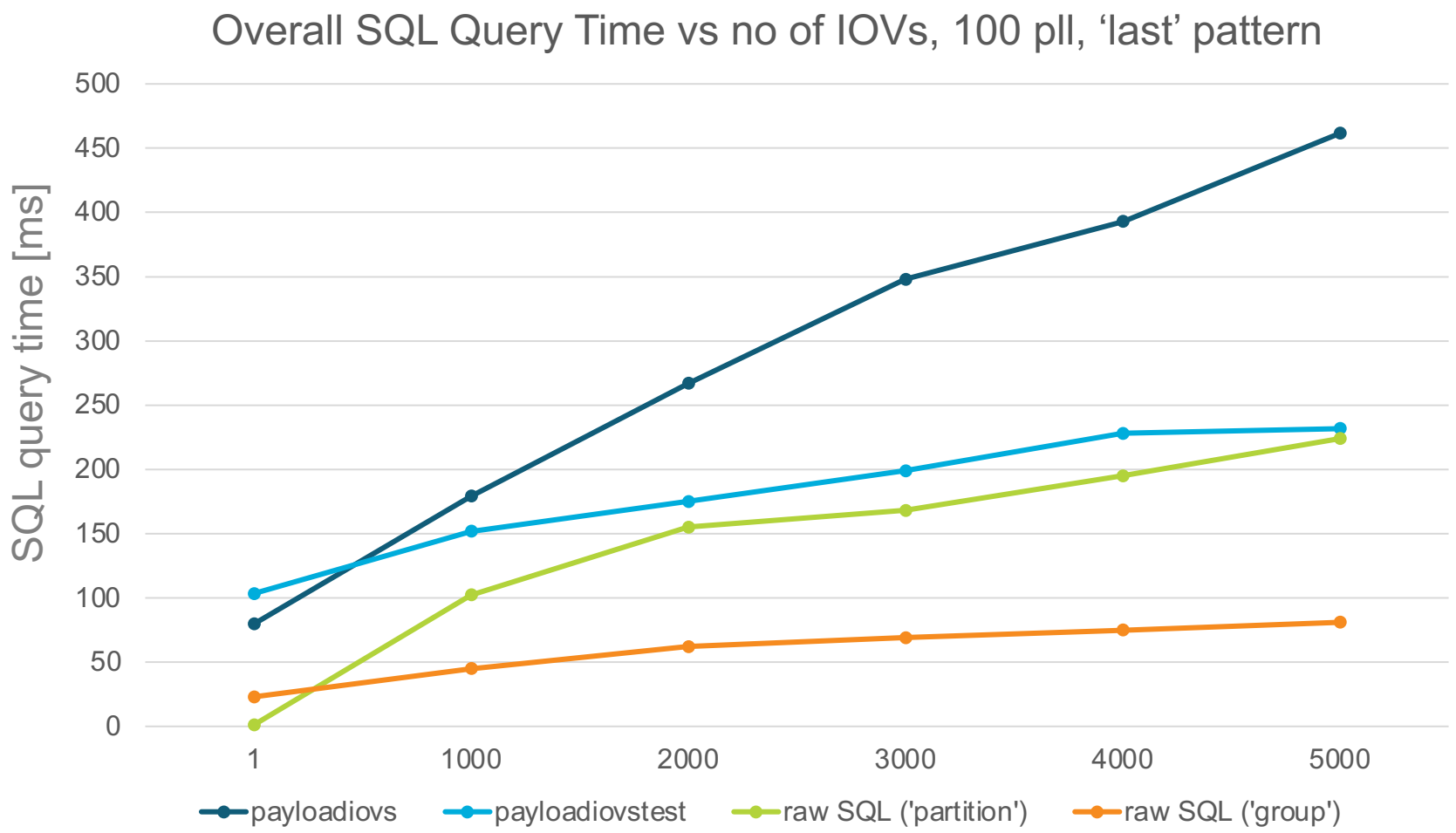
Full Raw SQL Query ‘lateral’

```
-- create new column
ALTER TABLE "PayloadIOV" ADD comb_iov NUMERIC(38,19);
UPDATE "PayloadIOV" SET comb_iov = major_iov + CAST(minor_iov AS DECIMAL(19,0)) / 10E18;

-- create covering index
CREATE INDEX covering_idx
ON "PayloadIOV" (payload_list_id, comb_iov DESC NULLS LAST);

-- actual query
SELECT pt.name AS payload_type_name, pi.payload_url, pi.major_iov, pi.minor_iov, pi.major_iov_end, pi.minor_iov_end
FROM "PayloadList" pl
JOIN "GlobalTag" gt ON pl.global_tag_id = gt.id AND gt.name = :'my_gt'
JOIN LATERAL (
   SELECT payload_url, major_iov, minor_iov, major_iov_end, minor_iov_end
   FROM   "PayloadIOV" pi
   WHERE  pi.payload_list_id = pl.id
     AND pi.comb_iov <= CAST(:my_major_iov + CAST(:my_minor_iov AS DECIMAL(19,0)) / 10E18 AS DECIMAL(38,19))
   ORDER BY pi.comb_iov DESC NULLS LAST
   LIMIT 1
) pi ON true
JOIN "PayloadType" pt ON pl.payload_type_id = pt.id;
```

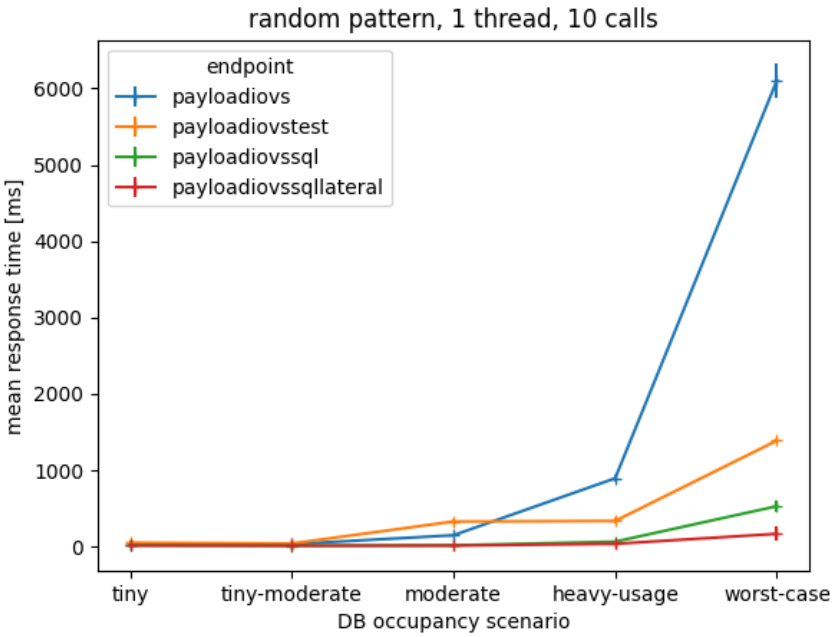
Django vs raw SQL – Performance

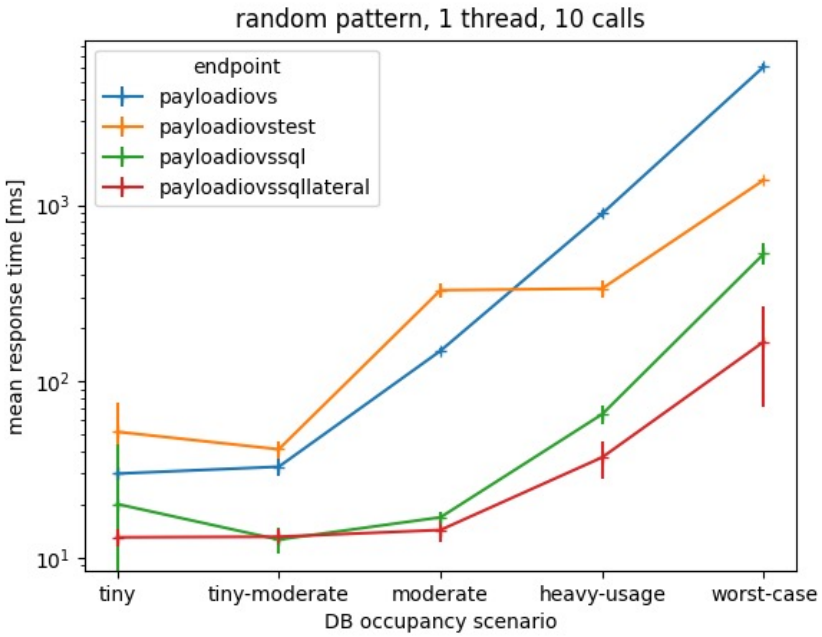


Django vs raw SQL – Performance



ORM vs Raw SQL - Endpoints

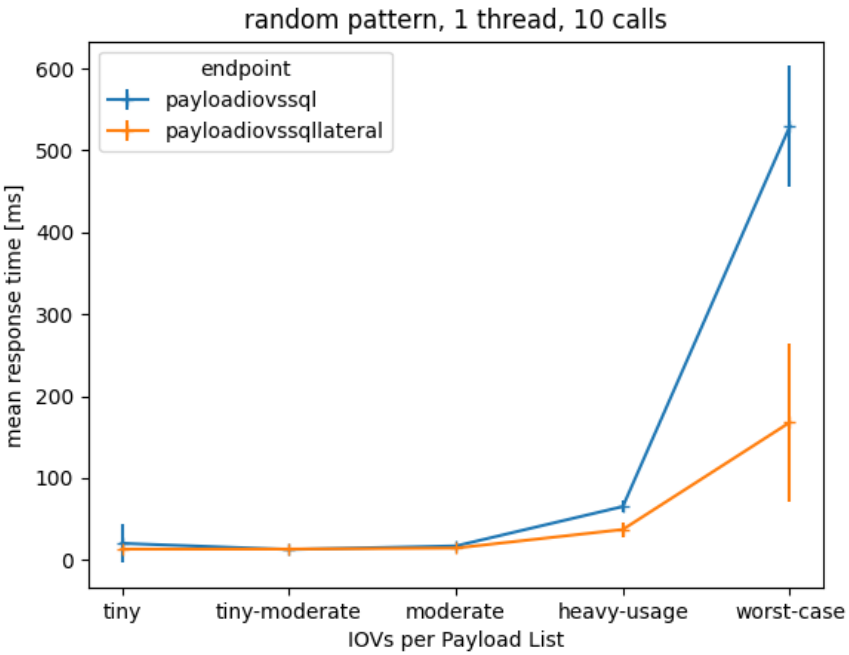


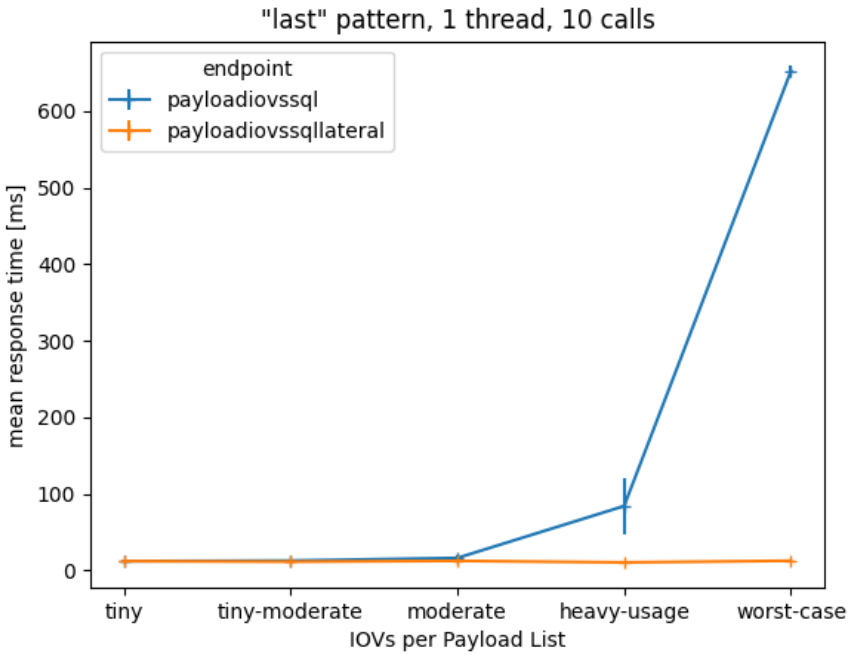



```
with connection.cursor() as cursor:
    cursor.execute(sql_string)
    row = cursor.fetchall()
return Response(row)
```

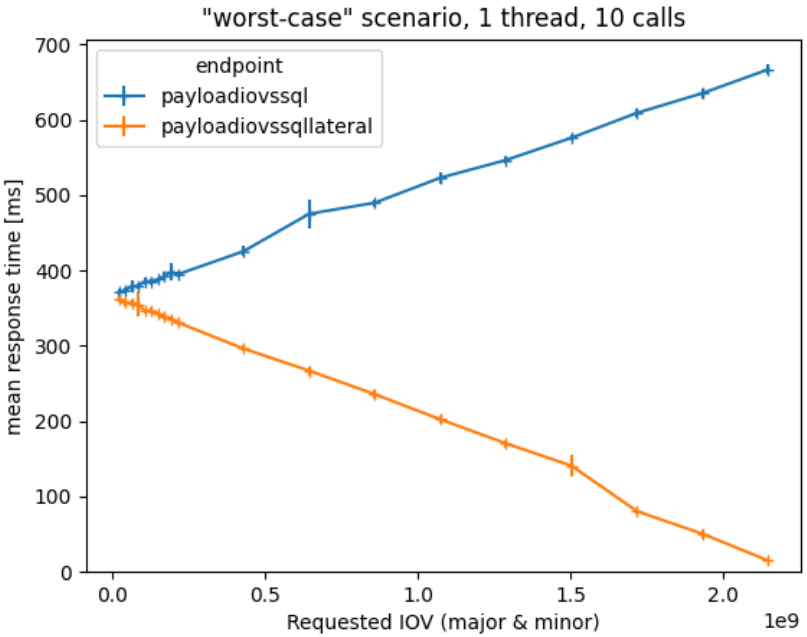
- Implement endpoints using raw SQL queries
  - Much faster than ORM (expected from query times)
- Focus on ‘lateral’ approach going forward

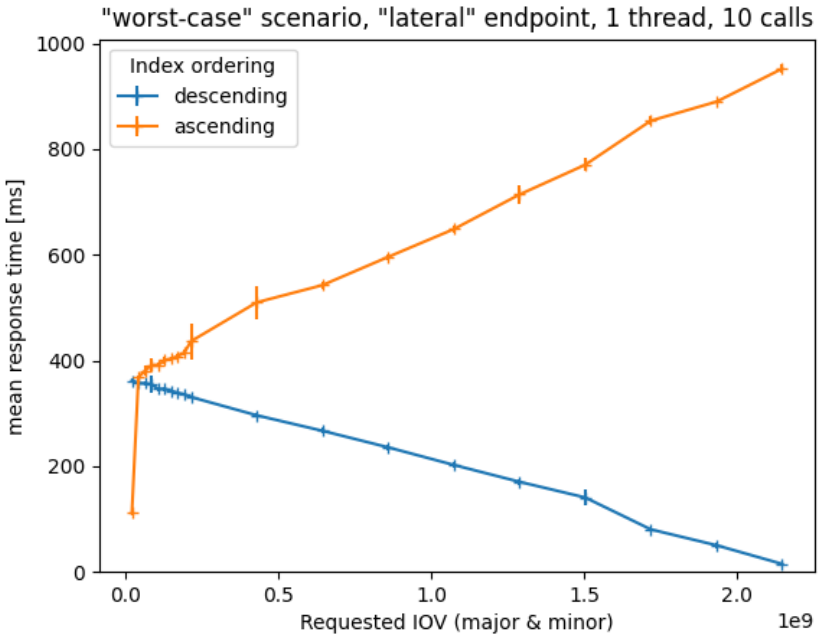
Raw SQL endpoints – ‘random’ vs ‘last'





Effect of Index Ordering





- 'lateral' always faster than 'groupby' approach
- Optimised for accessing last IOVs (descending index)
- Ascending index only faster for very small IOVs
- Descending Index generally better option (trying to find max)

Investigating Query Plans - I

```
Hash Join  (cost=7.23..410.15 rows=86 width=70) (actual time=6.111..365.158 rows=200 loops=1)
   Hash Cond: (pl.payload_type_id = pt.id)
   ->  Nested Loop  (cost=0.71..403.40 rows=86 width=69) (actual time=6.017..364.977 rows=200 loops=1)
         ->  Nested Loop  (cost=0.15..11.70 rows=86 width=16) (actual time=0.048..0.133 rows=201 loops=1)
               ->  Seq Scan on "GlobalTag" gt  (cost=0.00..1.09 rows=1 width=8) (actual time=0.023..0.025 rows=1 loops=1)
                     Filter: ((name)::text = 'worst-case'::text)
                     Rows Removed by Filter: 6
               ->  Index Scan using "PayloadList_global_tag_id_2b35c85f" on "PayloadList" pl
                           (cost=0.15..9.75 rows=86 width=24) (actual time=0.022..0.083 rows=201 loops=1)
                     Index Cond: (global_tag_id = gt.id)
         ->  Limit  (cost=0.56..4.53 rows=1 width=61) (actual time=1.815..1.815 rows=1 loops=201)
               ->  Index Only Scan using combo_covering_idx on "PayloadIOV" pi
                           (cost=0.56..3484.55 rows=876 width=61) (actual time=1.815..1.815 rows=1 loops=201)
                     Index Cond: (payload_list_id = pl.id)
                     Filter: ((major_iov < 100000000) OR ((major_iov = 100000000) AND (minor_iov <= 100000000)))
                     Rows Removed by Filter: 24669
                     Heap Fetches: 0
   ->  Hash  (cost=4.01..4.01 rows=201 width=17) (actual time=0.078..0.078 rows=201 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 19kB
         ->  Seq Scan on "PayloadType" pt  (cost=0.00..4.01 rows=201 width=17) (actual time=0.018..0.043 rows=201 loops=1)
 Planning Time: 0.996 ms
 Execution Time: 365.221 ms
```

major- & minorIOV

```
Hash Join  (cost=7.23..90.89 rows=86 width=70) (actual time=0.309..3.244 rows=200 loops=1)
   Hash Cond: (pl.payload_type_id = pt.id)
   ->  Nested Loop  (cost=0.71..84.14 rows=86 width=69) (actual time=0.075..2.935 rows=200 loops=1)
         ->  Nested Loop  (cost=0.15..11.70 rows=86 width=16) (actual time=0.028..0.121 rows=201 loops=1)
               ->  Seq Scan on "GlobalTag" gt  (cost=0.00..1.09 rows=1 width=8) (actual time=0.013..0.018 rows=1 loops=1)
                     Filter: ((name)::text = 'worst-case'::text)
                     Rows Removed by Filter: 6
               ->  Index Scan using "PayloadList_global_tag_id_2b35c85f" on "PayloadList" pl
                           (cost=0.15..9.75 rows=86 width=24) (actual time=0.012..0.063 rows=201 loops=1)
                     Index Cond: (global_tag_id = gt.id)
         ->  Limit  (cost=0.56..0.82 rows=1 width=61) (actual time=0.014..0.014 rows=1 loops=201)
               ->  Index Only Scan using combo_covering_idx on "PayloadIOV" pi
                           (cost=0.56..232.55 rows=876 width=61) (actual time=0.013..0.013 rows=1 loops=201)
                     Index Cond: ((payload_list_id = pl.id) AND (major_iov < 100000000))
                     Heap Fetches: 0
   ->  Hash  (cost=4.01..4.01 rows=201 width=17) (actual time=0.073..0.074 rows=201 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 19kB
         ->  Seq Scan on "PayloadType" pt  (cost=0.00..4.01 rows=201 width=17) (actual time=0.008..0.036 rows=201 loops=1)
 Planning Time: 0.645 ms
 Execution Time: 3.299 ms
```

Only majorIOV

Investigating Query Plans - II

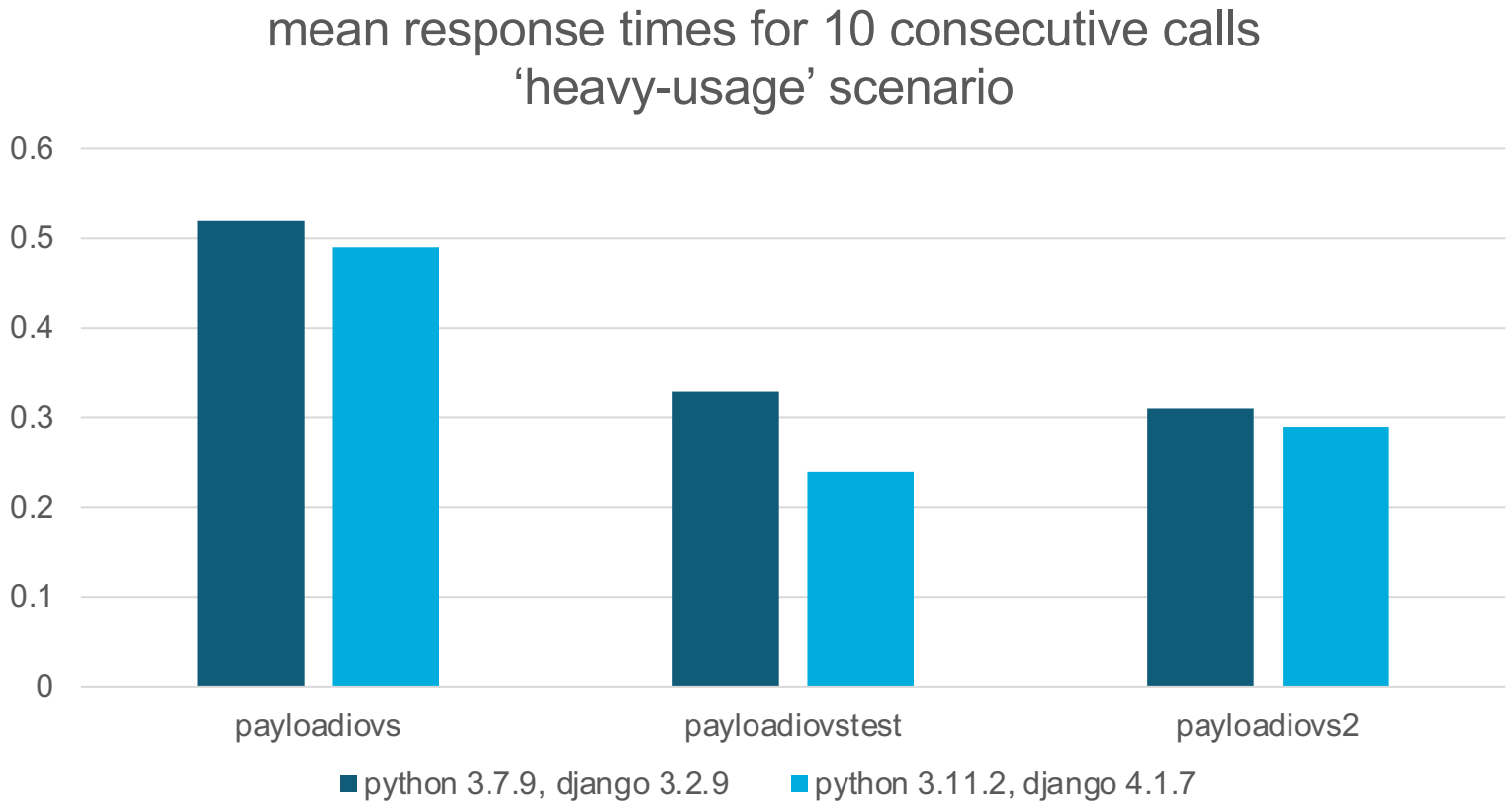
```
->  Limit  (cost=0.56..4.53 rows=1 width=61) (actual time=1.815..1.815 rows=1 loops=201)
      ->  Index Only Scan using combo_covering_idx on "PayloadIOV" pi
                  (cost=0.56..3484.55 rows=876 width=61) (actual time=1.815..1.815 rows=1 loops=201)
            Index Cond: (payload_list_id = pl.id)
            Filter: ((major_iov < 100000000) OR ((major_iov = 100000000) AND (minor_iov <= 100000000)))
            Rows Removed by Filter: 24669
            Heap Fetches: 0
```

Index Condition & Filter

```
->  Limit  (cost=0.56..0.82 rows=1 width=61) (actual time=0.014..0.014 rows=1 loops=201)
      ->  Index Only Scan using combo_covering_idx on "PayloadIOV" pi
                  (cost=0.56..232.55 rows=876 width=61) (actual time=0.013..0.013 rows=1 loops=201)
            Index Cond: ((payload_list_id = pl.id) AND (major_iov < 100000000))
            Heap Fetches: 0
```

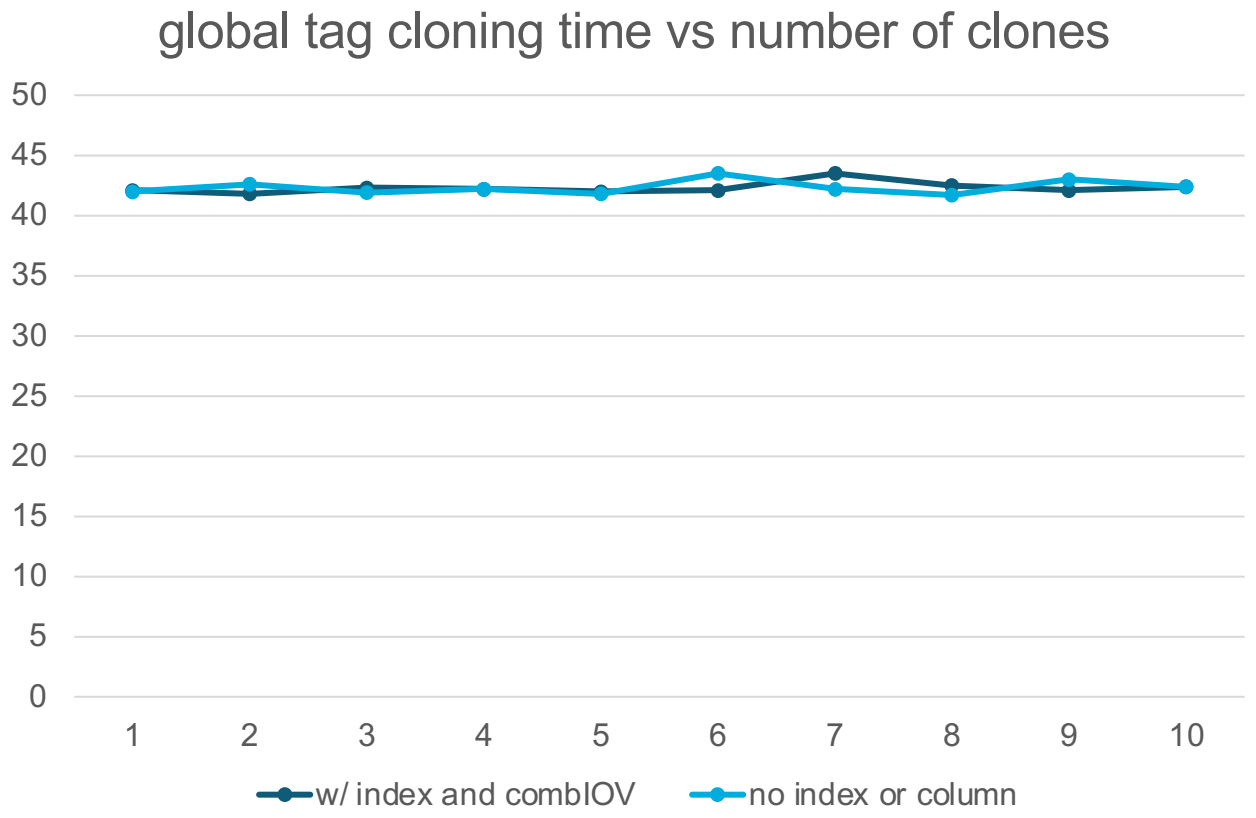
Index Condition Only

Python / Django Versions



- Later python & Django versions improve performance
  - Especially in for endpoint ‘payloadiovstest’ (0.33s vs 0.24s)

Global Tag Cloning - Performance



- New index and column have no significant impact on cloning time