Increased I/O Observability with `pg_stat_io`

Postgres Performance Observability Sources and Analysis Techniques

Melanie Plageman

@Microsoft

- Open source Postgres hacking: executor, planner, storage, and statistics sub-systems
- I/O Benchmarking and Linux kernel storage performance tuning
- Recently worked on prefetching for direct I/O and I/O statistics

https://github.com/melanieplageman

Transactional Workload I/O Performance Goals

High transactions per second (TPS)

Consistent low latency

Common I/O Performance Issue Causes

- Working set is not in memory
- Spikey checkpoint I/O
- Autovacuum frequency too low

Postgres I/O Tuning Targets

Shared buffers

Background writer

Checkpointer

Autovacuum

Postgres I/O Statistics Views

`pg_stat_database`

- hits, reads, read time, write time

`pg_statio_all_tables`

- hits, reads

`pg_stat_bgwriter`

- backend writes, backend fsyncs

`pg_stat_statements`

- shared and local buffer hits, reads, writes, read time, write time

Gaps in Postgres I/O Statistics Views

- ***Writes*** = flushes + ***extends***
- ***Reads*** and ***writes*** combined for all ***backend types***
- I/O combined for all ***contexts*** and on all ***objects***

backend_type	io_object	io_context	reads	read_time	writes	write_time	extends	extend_time	op_bytes	hits	evictions	reuses	fsyncs	fsync_time
autovacuum launcher	relation	bulkread	0	0	0	0			8192	0	0	0		
autovacuum launcher	relation	normal	3	0.022	0	0			8192	3	0		0	0
autovacuum worker	relation	bulkread	0	0	0	0			8192	0	0	0		
autovacuum worker	relation	normal	435	5.703	0	0	10	0.445	8192	13305	0		0	0
autovacuum worker	relation	vacuum	156	0.756	0	0	0	0	8192	1858	0	94		
client backend	relation	bulkread	893	0	0	0			8192	14	0	131		
client backend	relation	bulkwrite	0	0	0	0	0	0	8192	0	0	0		
client backend	relation	normal	412	4.455	0	0	888	32.932	8192	210308	0		0	0
client backend	relation	vacuum	0	0	0	0	0	0	8192	0	0	0		
client backend	temp relation	normal	0	0	0	0	0	0	8192	0	0			
background worker	relation	bulkread	0	0	0	0			8192	0	0	0		
background worker	relation	bulkwrite	0	0	0	0	0	0	8192	0	0	0		
background worker	relation	normal	0	0	0	0	0	0	8192	0	0		0	0
background worker	relation	vacuum	0	0	0	0	0	0	8192	0	0	0		
background worker	temp relation	normal	0	0	0	0	0	0	8192	0	0			
background writer	relation	normal			0	0			8192				0	0
checkpointer	relation	normal			2398	42.673			8192				283	577.582
standalone backend	relation	bulkread	0	0	0	0			8192	0	0	0		
standalone backend	relation	bulkwrite	0	0	0	0	8	0	8192	7	0	0		
standalone backend	relation	normal	536	0	987	0	640	0	8192	76051	0		0	0
standalone backend	relation	vacuum	10	0	0	0	0	0	8192	877	0	0		
startup	relation	bulkread	0	0	0	0			8192	0	0	0		
startup	relation	bulkwrite	0	0	0	0	0	0	8192	0	0	0		

pg_stat_io (PG 16)

- backend_type, io_object, io_context
- reads*, writes*, extends*, op_bytes, hits, evictions, reuses, fsyncs*

Why Count Flushes and Extends Separately?

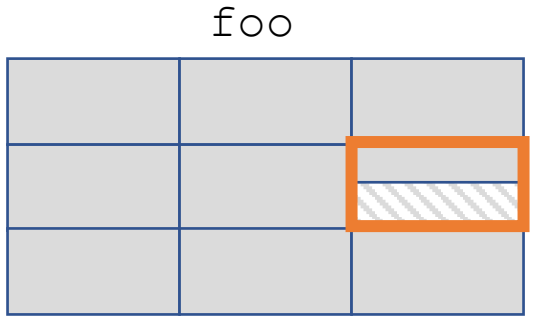
`pg_stat_io`

- ***write*** = flush
- ***extend*** = extend

Postgres UPDATE/INSERT I/O Workflow

```
INSERT INTO foo VALUES(1,1);
```

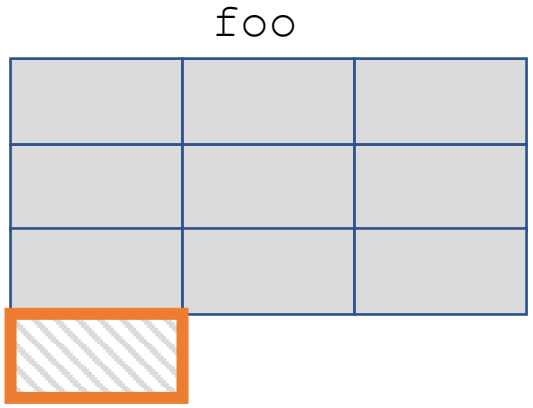
1. Find a disk block with enough space to fit the new data



Postgres UPDATE/INSERT I/O Workflow

```
INSERT INTO foo VALUES(1,1);
```

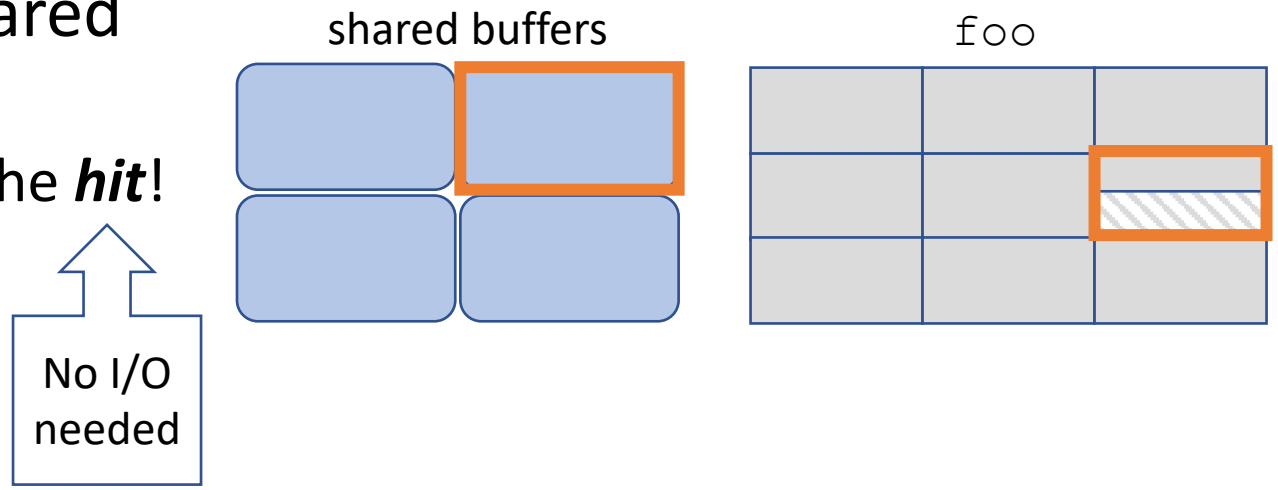
1. Find a disk block with enough space to fit the new data
   i. If no block has enough free space, ***extend*** the file.



Postgres UPDATE/INSERT I/O Workflow

```
INSERT INTO foo VALUES(1,1);
```

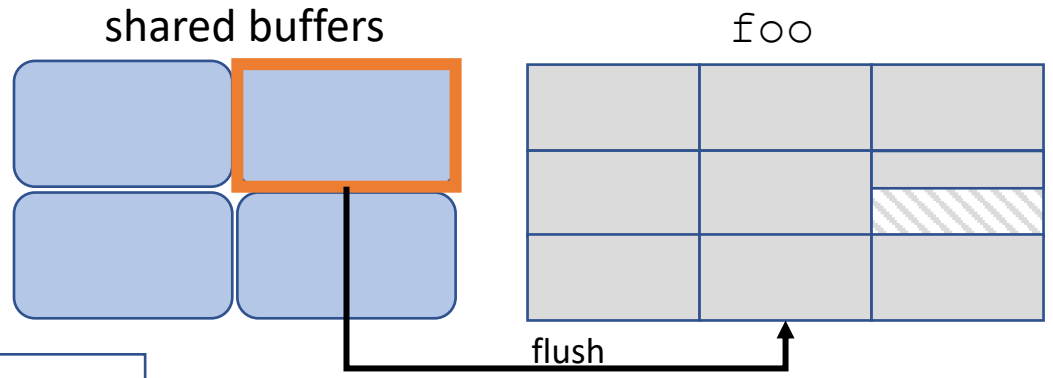
1. Find a disk block with enough space to fit the new data
   i. If no block has enough free space, ***extend*** the file.
2. Check for the block in shared buffers.
   i. If it is already loaded, cache ***hit***!



Postgres UPDATE/INSERT I/O Workflow

```
INSERT INTO foo VALUES(1,1);
```

1. Find a disk block with enough space to fit the new data
   i. If no block has enough free space, ***extend*** the file.
2. Check for the block in shared buffers.
   i. If it is already loaded, success!
3. Otherwise, find a shared buffer we can use.
   i. If it is dirty, ***flush*** it.

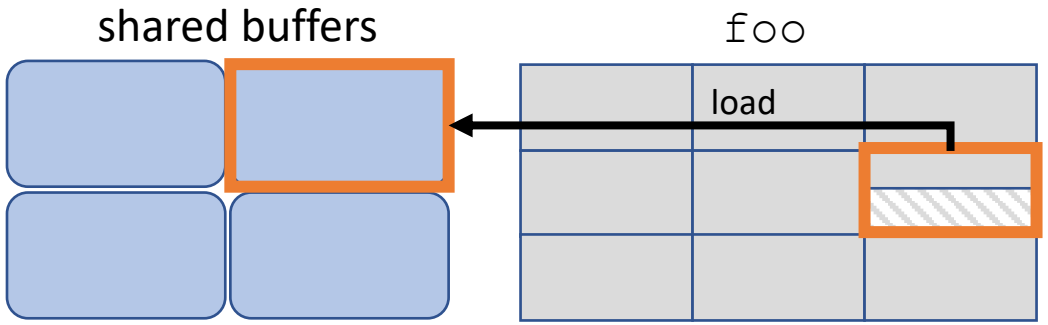


Flush = “write” in pg_stat_io

Postgres UPDATE/INSERT I/O Workflow

```
INSERT INTO foo VALUES(1,1);
```

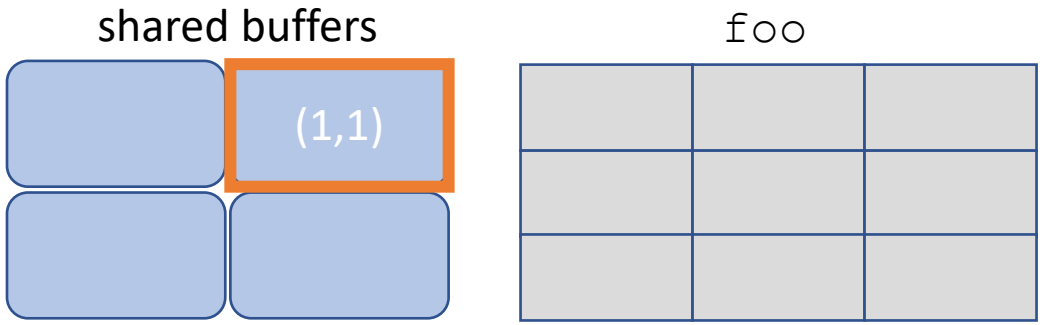
1. Find a disk block with enough space to fit the new data
   i. If no block has enough free space, ***extend*** the file.
2. Check for the block in shared buffers.
   i. If it is already loaded, success!
3. Otherwise, find a shared buffer we can use.
   i. If it is dirty, flush it.
4. ***Read*** our block into the buffer.



Postgres UPDATE/INSERT I/O Workflow

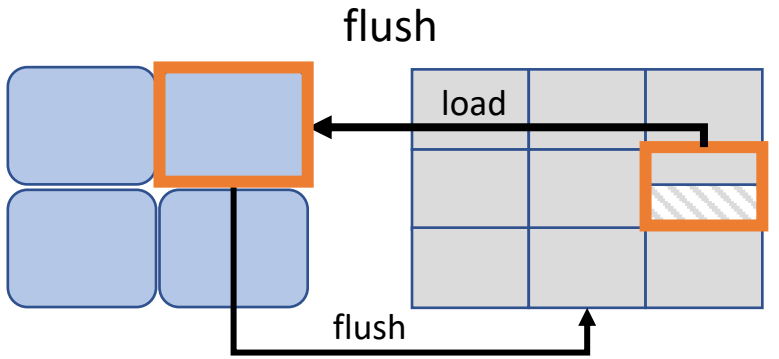
```
INSERT INTO foo VALUES(1,1);
```

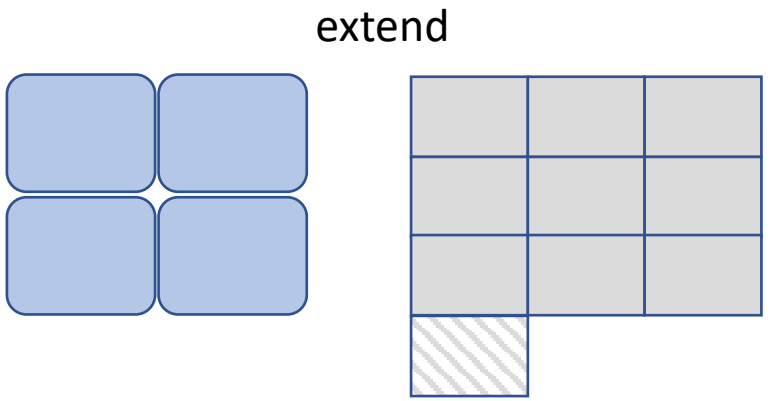
1. Find a disk block with enough space to fit the new data
   i. If no block has enough free space, ***extend*** the file.
2. Check for the block in shared buffers.
   i. If it is already loaded, success!
3. Otherwise, find a shared buffer we can use.
   i. If it is dirty, flush it.
4. Read our block into the buffer.
5. Write our data into the buffer.



Why Count Flushes and Extends Separately?

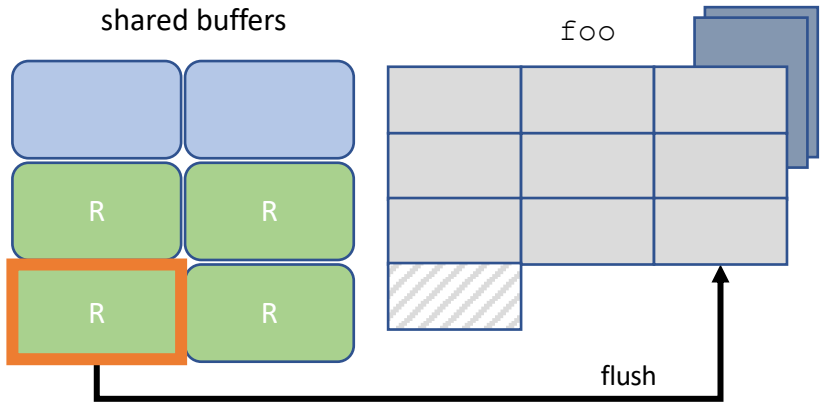
- Synchronous flushes are avoidable
- Extends are unavoidable
- Separating them allows tuning discretion





Why Count Flushes and Extends Separately?

- Extends are normal for bulk writes
- COPY FROM does lots of extends
- Data loaded may not be part of transactional workload working set



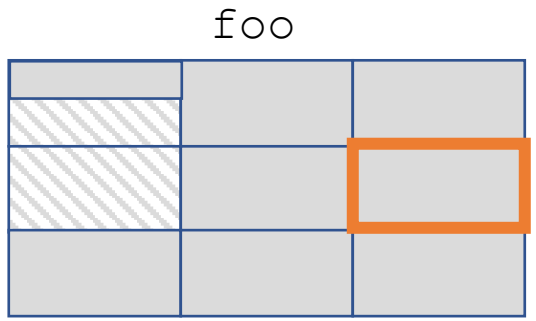
Why Track I/O Per Context or Per Backend Type?

`pg_stat_io`

- ***backend_type***
- ***io_context***

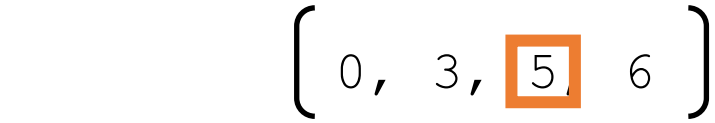
Postgres Autovacuum Workflow

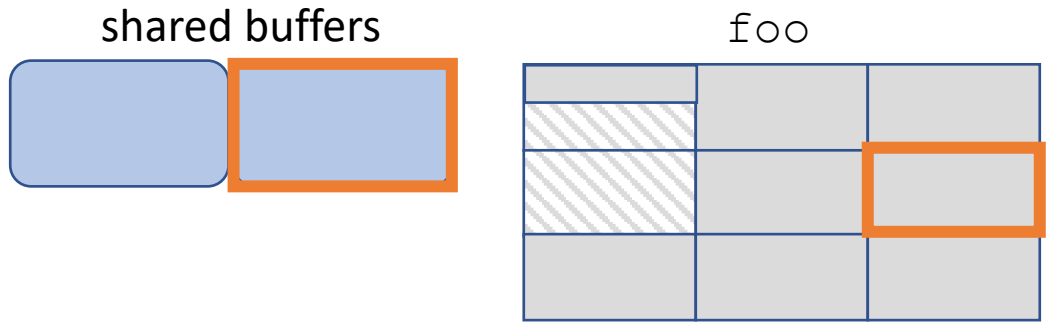
1. Identify the next block to vacuum.



Postgres Autovacuum Workflow

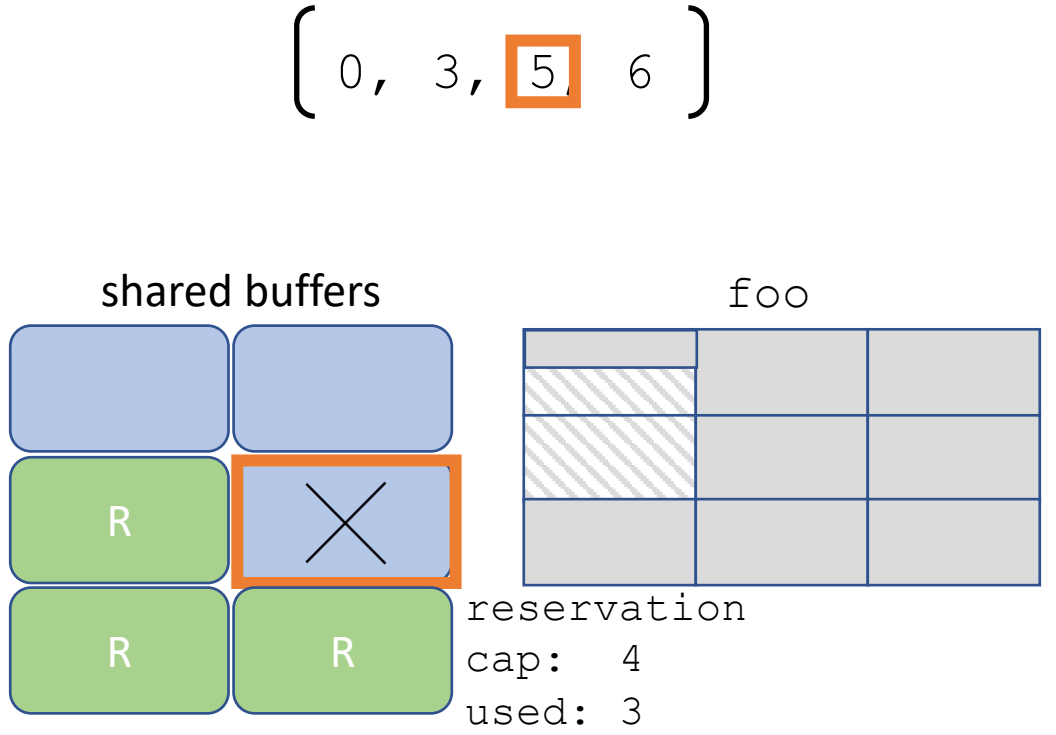
1. Identify the next block to vacuum.
2. Check for the block in shared buffers.
   i. If it is, vacuum it! (cache ***hit***)





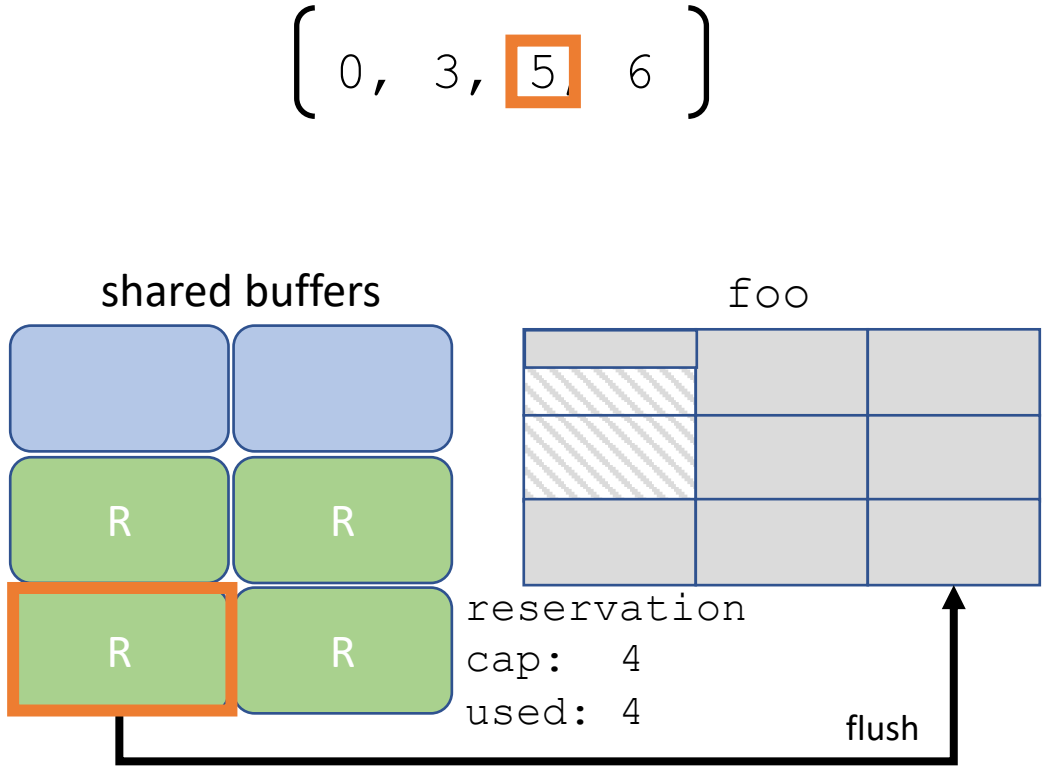
Postgres Autovacuum Workflow

1. Identify the next block to vacuum.
2. Check for the block in shared buffers.
   i. If it is, vacuum it!
3. Otherwise, find the next reserved buffer to use.
   i. If we are not at the reservation cap, ***evict*** a shared buffer.



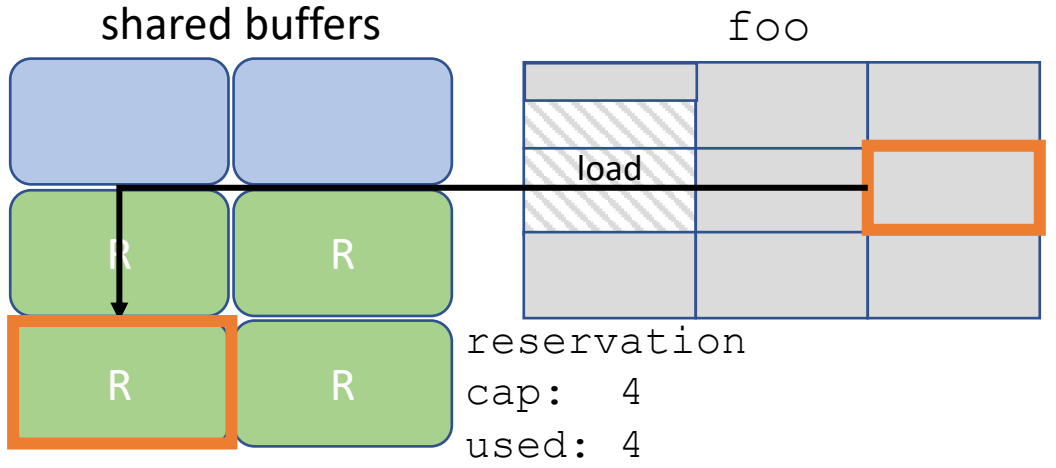
Postgres Autovacuum Workflow

1. Identify the next block to vacuum.
2. Check for the block in shared buffers.
   i. If it is, vacuum it!
3. Otherwise, find the next reserved buffer to use.
   i. If we are not at the reservation cap, evict a shared buffer.
   ii. If we are ***reusing*** a dirty, reserved buffer, ***flush*** it.



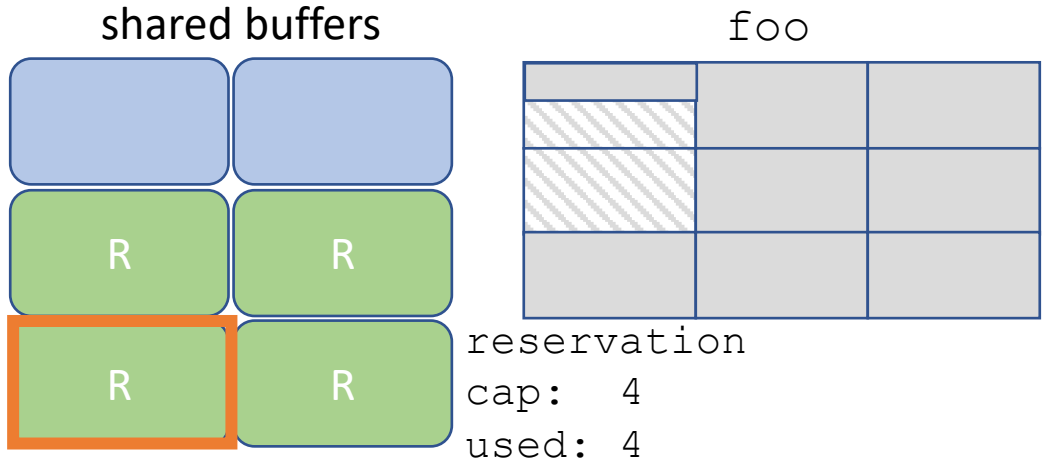
Postgres Autovacuum Workflow

1. Identify the next block to vacuum.
2. Check for the block in shared buffers.
   i. If it is, vacuum it!
3. Find the next reserved buffer to use.
   i. If we are not at the reservation cap, evict a shared buffer.
   ii. If we are reusing a dirty, reserved buffer, flush it.
4. ***Read*** the block into the buffer.



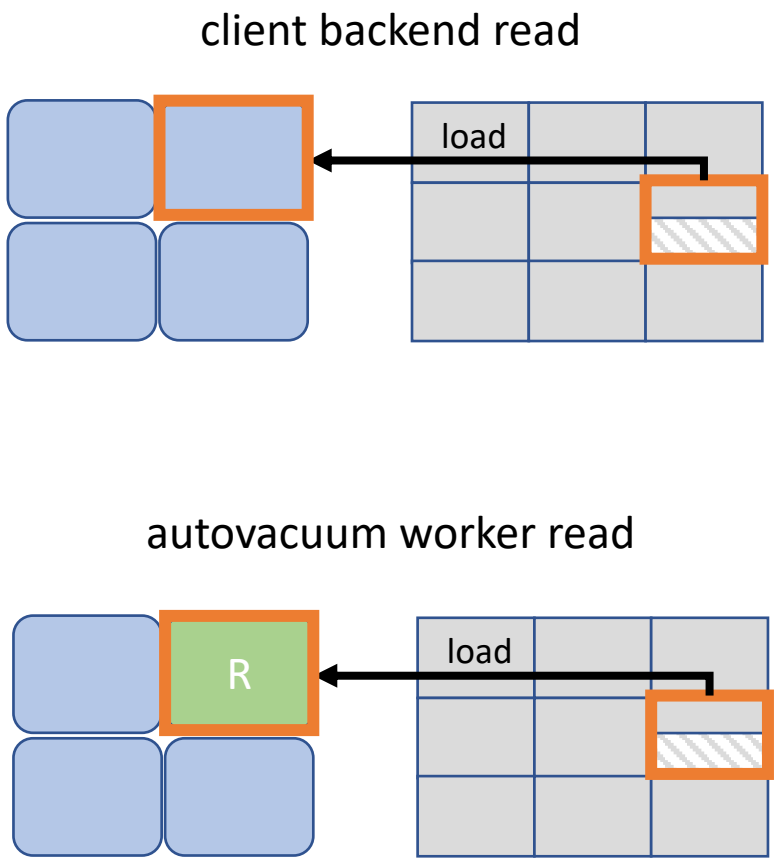
Postgres Autovacuum Workflow

1. Identify the next block to vacuum.
2. Check for the block in shared buffers.
   i. If it is, vacuum it!
3. Find the next reserved buffer to use.
   i. If we are not at the reservation cap, evict a shared buffer.
   ii. If we are reusing a dirty, reserved buffer, flush it.
4. Read the block into the buffer.
5. Vacuum the buffer and mark it dirty.



Why Track I/O Per Backend Type?

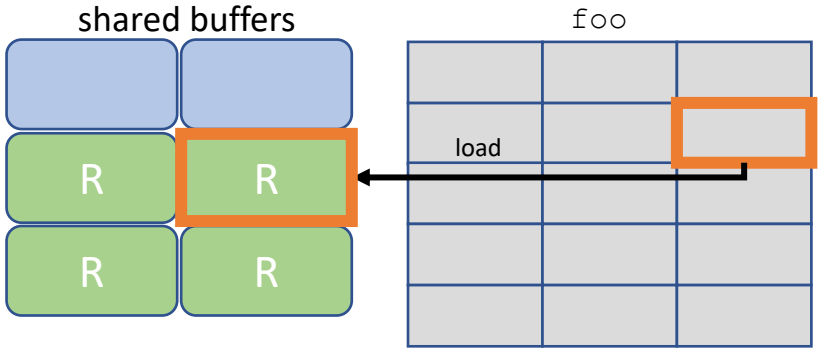
- Not all I/O is for blocks that are part of the working set
- Autovacuum worker reads are often of older data



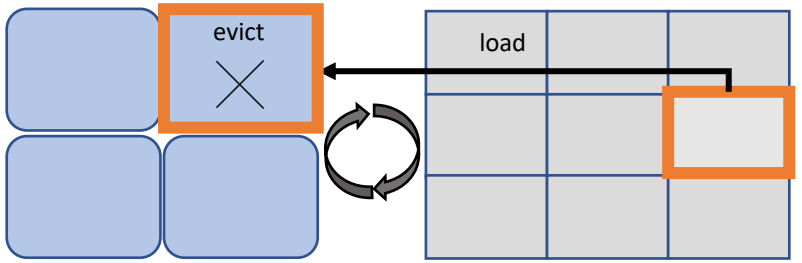
Why Track I/O Per Context?

- High number of reads during bulk read operations of data not in shared buffers.
- Shared buffers not used for all I/O
- Large* SELECTs not in shared buffers

large SELECT (bulkread context) read



client backend normal context cache miss



*large = table blocks > shared buffers / 4

Data-Driven Tuning with pg_stat_io

```
   backend_type    | io_object | io_context |  reads   |   hits    |  writes
-------------------+-----------+------------+----------+-----------+----------
 client backend    | relation  | normal     | 99844392 | 321091023 | 18120752
 background writer | relation  | normal     |          |           | 20100354
 checkpointer      | relation  | normal     |          |           |   753214
(3 rows)
```

Background Writer Too Passive

- client backend normal context writes high
- background writer normal context writes high
- checkpointer writes lower than bgwriter

```
   backend_type    | io_object | io_context |   reads   |   hits    |  writes   | evictions
-------------------+-----------+------------+-----------+-----------+-----------+-----------
 client backend    | relation  | normal     | 128443922 | 201091002 | 128143381 | 127318729
 client backend    | relation  | vacuum     |     86507 |      5906 |     86442 |       129
 autovacuum worker | relation  | vacuum     |   6920602 |    492178 |   6911103 |      2457
(3 rows)
```

Shared Buffers Too Small

- client backend normal context reads high
- evictions high
- client backend writes/read ≈ 1
- cache hit ratio ≈ 60%

Cache hit ratio query

```
SELECT (hits / (reads + hits)::float) * 100
  FROM pg_stat_io
  WHERE backend_type = 'client backend' AND
    io_object = 'relation' AND
    io_context = 'normal';
```

backend_type	io_object	io_context	reads	hits	evictions
client backend	relation	normal	129121	299627939	6613
client backend	relation	vacuum	6886	4598	24
autovacuum worker	relation	vacuum	9988377	8910202	3115
autovacuum worker	relation	normal	778	67140	11
client backend	relation	bulkread	359631871	126	33

(5 rows)

Shared Buffers Not Too Small

- client backend normal context reads low
- client backend bulkread context reads high
- vacuum reads high

Calculating accurate cache hit ratios

pg_stat_database

```
SELECT
(sum(blks_hit) /
(sum(blks_read) + sum(blks_hit))::float) * 100
FROM pg_stat_database;
```

45%

```
 all_blks_hit | all_blks_read
--------------+---------------
    308610005 |     369757033
(1 row)
```

pg_stat_io

```
SELECT (hits/(reads + hits)::float) * 100
FROM pg_stat_io;
```

45%

```
   backend_type    | io_object | io_context |   reads   |   hits    | evictions
-------------------+-----------+------------+-----------+-----------+-----------
 client backend    | relation  | normal     |    129121 | 299627939 |      6613
 client backend    | relation  | vacuum     |      6886 |      4598 |        24
 autovacuum worker | relation  | vacuum     |   9988377 |   8910202 |      3115
 autovacuum worker | relation  | normal     |       778 |     67140 |        11
 client backend    | relation  | bulkread   | 359631871 |       126 |        33
(5 rows)
```

Calculating accurate cache hit ratios

pg_stat_database

```
SELECT
(sum(blks_hit) /
(sum(blks_read) + sum(blks_hit))::float) * 100
FROM pg_stat_database;
```

45%

```
 all_blks_hit | all_blks_read
--------------+---------------
    308610005 |     369757033
(1 row)
```

pg_stat_io

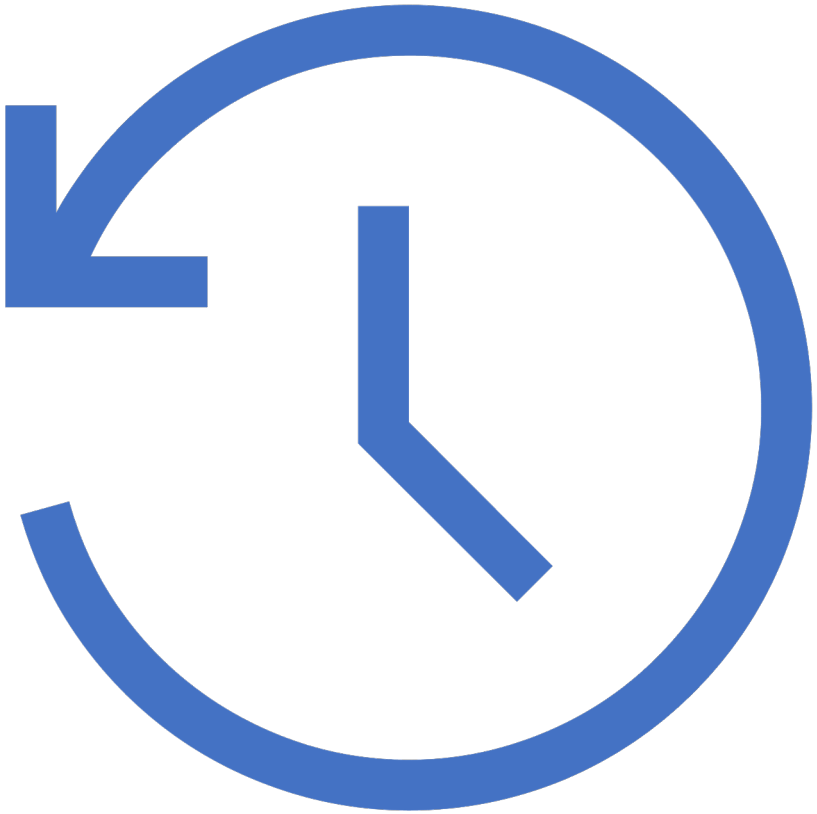
```
SELECT (hits/(reads + hits)::float) * 100
FROM pg_stat_io
WHERE backend_type = 'client backend'
  AND io_object = 'relation'
  AND io_context = 'normal';
```

99.9%

```
   backend_type    | io_object | io_context |   reads   |   hits    | evictions
-------------------+-----------+------------+-----------+-----------+-----------
 client backend    | relation  | normal     |    129121 | 299627939 |      6613
 client backend    | relation  | vacuum     |      6886 |      4598 |        24
 autovacuum worker | relation  | vacuum     |   9988377 |   8910202 |      3115
 autovacuum worker | relation  | normal     |       778 |     67140 |        11
 client backend    | relation  | bulkread   | 359631871 |       126 |        33
(5 rows)
```

Future additions

- “bypass” IO
- per connection IO stats
- consolidated WAL stats



Contact me:
@melanieplageman