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

Autovacuum bottlenecked on I/O

Postgres I/O Tuning Targets

Shared buffers

Background writer

Autovacuum

Postgres I/O Statistics Views

`pg_stat_database`
- hits, reads

`pg_statio_all_tables`
- hits, reads, read time, write time

`pg_stat_bgwriter`
- backend writes, backend fsyncs

`pg_stat_statements`
- shared buffer hits, reads, writes, read time, write time
- local buffer hits, reads, writes, read time, write time

Postgres I/O Statistics Views' Gaps

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

pg_stat_io (pg 16)

backend_type	io_object	io_context	reads	writes	extends	op_bytes	evictions	reuses	fsyncs
autovacuum launcher	relation	bulkread	0	0		8192	0	0	
autovacuum launcher	relation	normal	1	0		8192	0		0
autovacuum worker	relation	bulkread	0	0		8192	0	0	
autovacuum worker	relation	normal	174	0	11	8192	0		0
autovacuum worker	relation	vacuum	125	0	0	8192	0	93	
client backend	relation	bulkread	891	0		8192	0	130	
client backend	relation	bulkwrite	891	0	0	8192	0	0	
client backend	relation	normal	191	0	0	8192	0		0
client backend	relation	vacuum	0	0	0	8192	0	0	
client backend	temp relation	normal	0	0	0	8192	0		
background worker	relation	bulkread	0	0		8192	0	0	
background worker	relation	bulkwrite	0	0	0	8192	0	0	
background worker	relation	normal	0	0	0	8192	0		0
background worker	relation	vacuum	0	0	0	8192	0	0	
background worker	temp relation	normal	0	0	0	8192	0		
background writer	relation	normal		0		8192			0
checkpointer	relation	normal		894		8192			248
standalone backend	relation	bulkread	0	0		8192	0	0	
standalone backend	relation	bulkwrite	0	0	8	8192	0	0	
standalone backend	relation	normal	689	983	470	8192	0		0
standalone backend	relation	vacuum	10	0	0	8192	0	0	
startup	relation	bulkread	0	0		8192	0	0	
startup	relation	bulkwrite	0	0	0	8192	0	0	
startup	relation	normal	0	0	0	8192	0		0
startup	relation	vacuum	0	0	0	8192	0	0	
walsender	relation	bulkread	0	0		8192	0	0	
walsender	relation	bulkwrite	0	0	0	8192	0	0	
walsender	relation	normal	0	0	0	8192	0		0
walsender	relation	vacuum	0	0	0	8192	0	0	
walsender	temp relation	normal	0	0	0	8192	0		

backend_type, io_object, io_context,

reads, writes, extends, evictions, reuses, fsyncs

Why Count Flushes and Extends Separately?

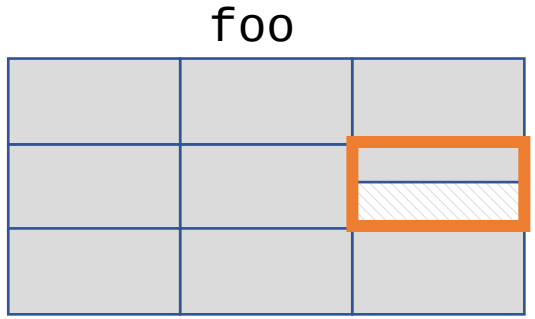
`pg_stat_io`

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

Postgres UPDATE/INSERT I/O Workflow

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

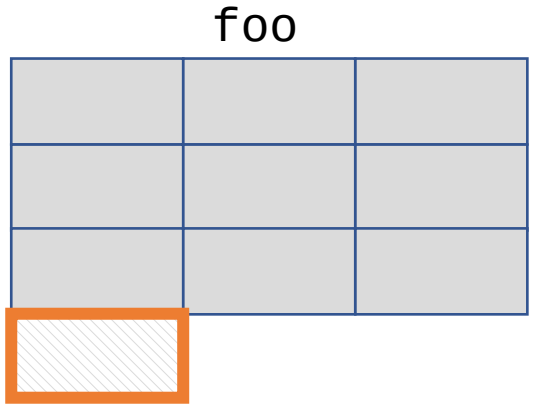
1. Find a disk block with enough space to fit the new data



Postgres UPDATE/INSERT I/O Workflow

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

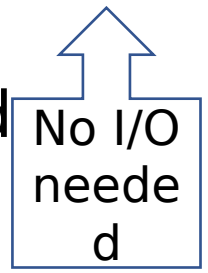
1. Find a disk block with enough space to fit the new data
   i. If no block has enough free space, ***extend*** the file.

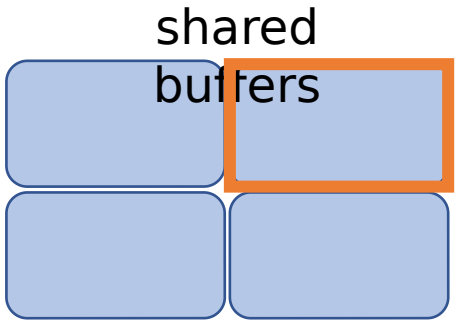


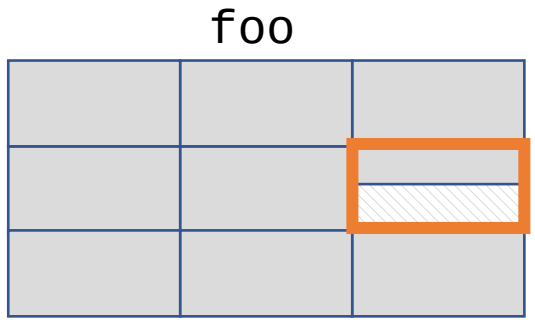
Postgres UPDATE/INSERT I/O Workflow

```
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***!



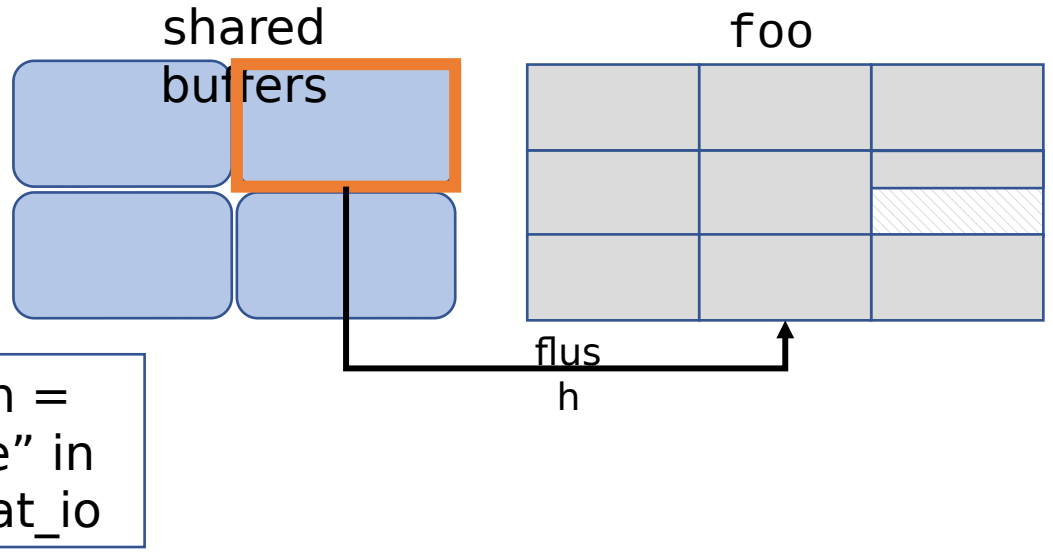




Postgres UPDATE/INSERT I/O Workflow

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

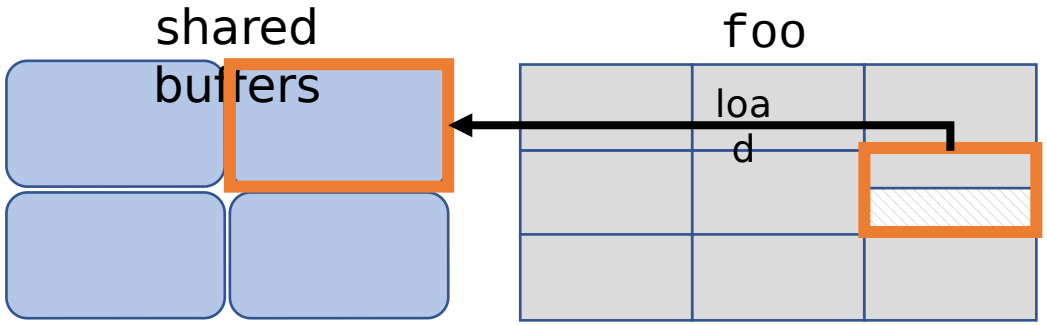
1. Find a disk block with enough space to fit the new data
   i. If no block has enough free space, ***extend*** the file.
2. Check for the block in shared buffers.
   i. If it is already loaded, success!
3. Otherwise, find a shared buffer we can use.
   i. If it is dirty, ***flush*** it.



Postgres UPDATE/INSERT I/O Workflow

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

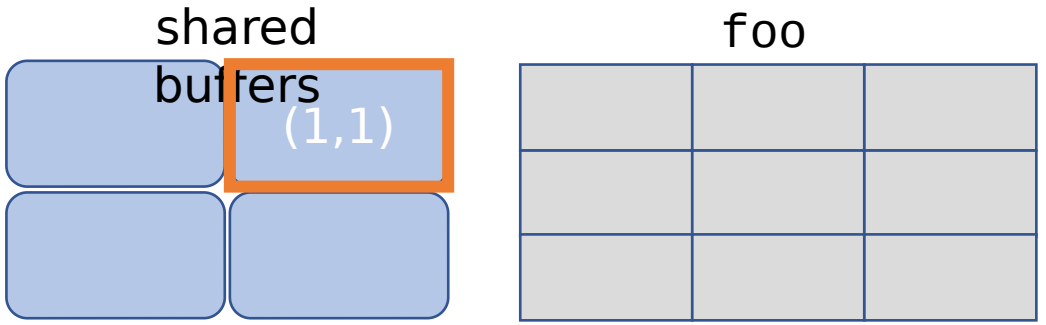
1. Find a disk block with enough space to fit the new data
   i. If no block has enough free space, ***extend*** the file.
2. Check for the block in shared buffers.
   i. If it is already loaded, success!
3. Otherwise, find a shared buffer we can use.
   i. If it is dirty, flush it.
4. ***Read*** our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



Postgres UPDATE/INSERT I/O Workflow

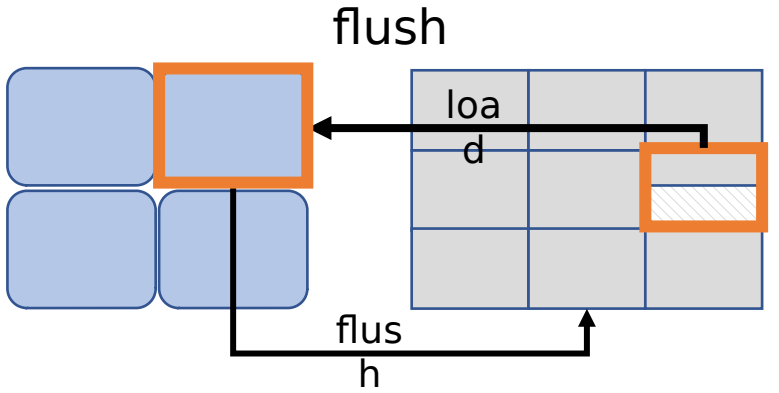
```
INSERT INTO foo VALUES(1,1);
```

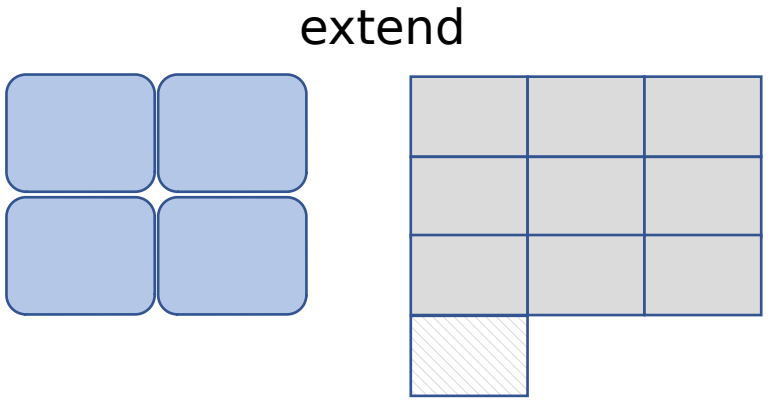
1. Find a disk block with enough space to fit the new data
   i. If no block has enough free space, ***extend*** the file.
2. Check for the block in shared buffers.
   i. If it is already loaded, success!
3. Otherwise, find a shared buffer we can use.
   i. If it is dirty, flush it.
4. Read our block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



Why Count Flushes and Extends Separately?

- Synchronous flushes are avoidable





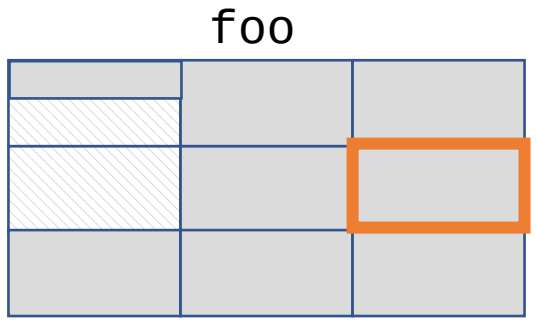
Why Track I/O Per Context or Per Backend Type?

`pg_stat_io`

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

Postgres Autovacuum Workflow

1. Identify the next block to vacuum.

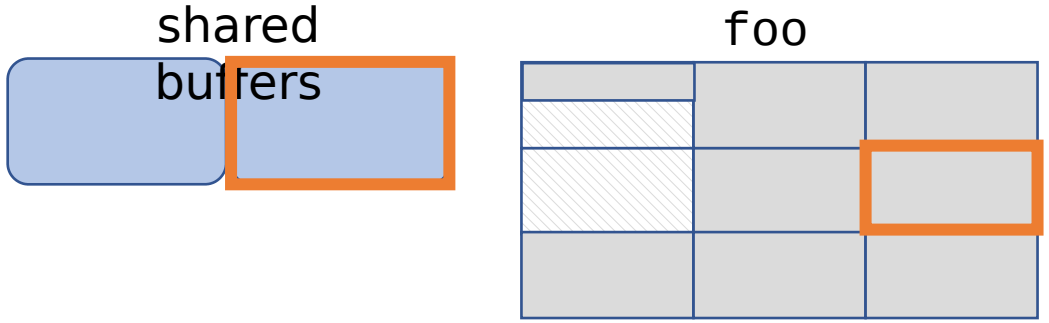


Postgres Autovacuum Workflow

1. Identify the next block to vacuum.

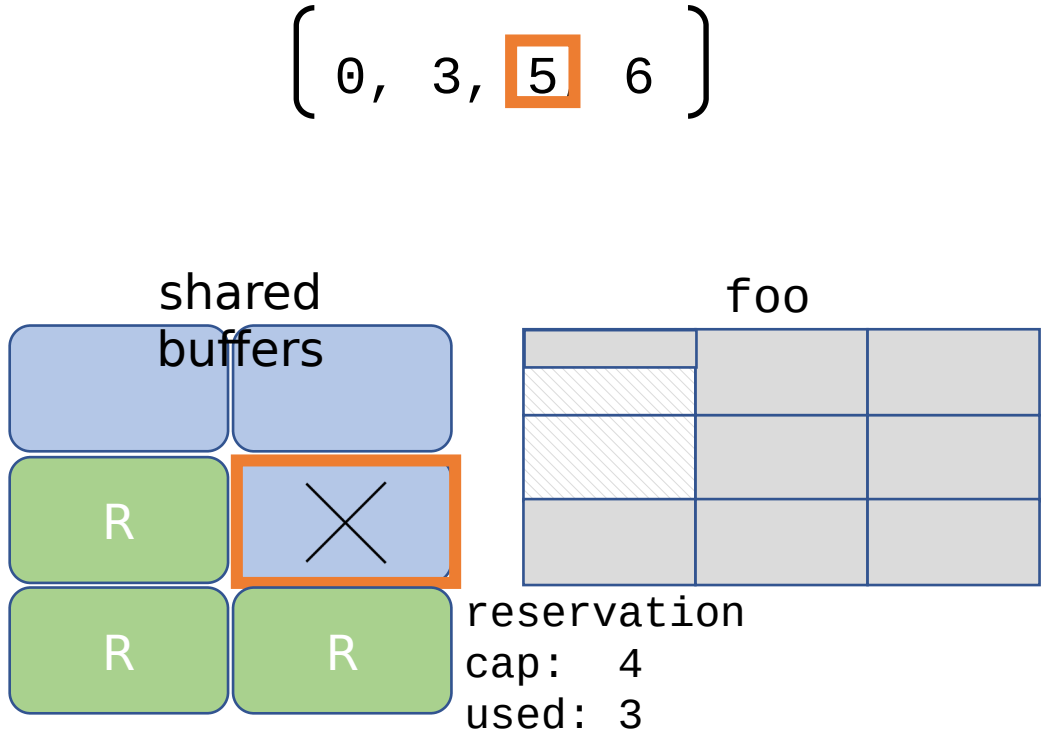
```
( 0, 3, 5, 6 )
```

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



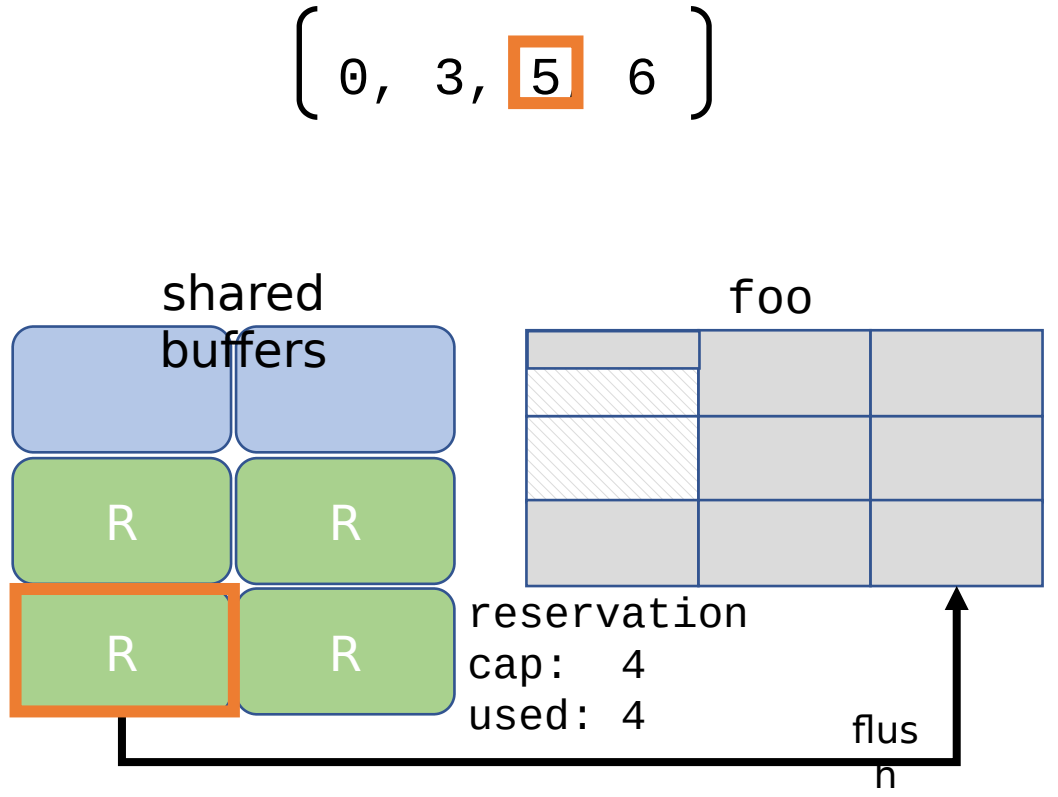
Postgres Autovacuum Workflow

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.



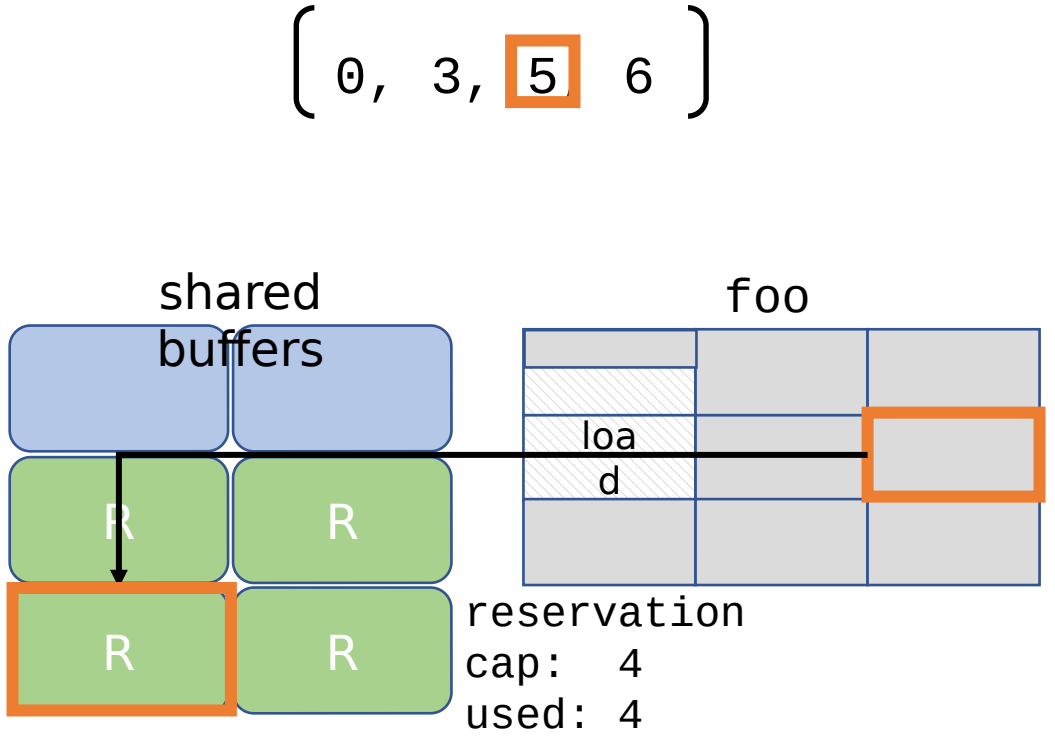
Postgres Autovacuum Workflow

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.



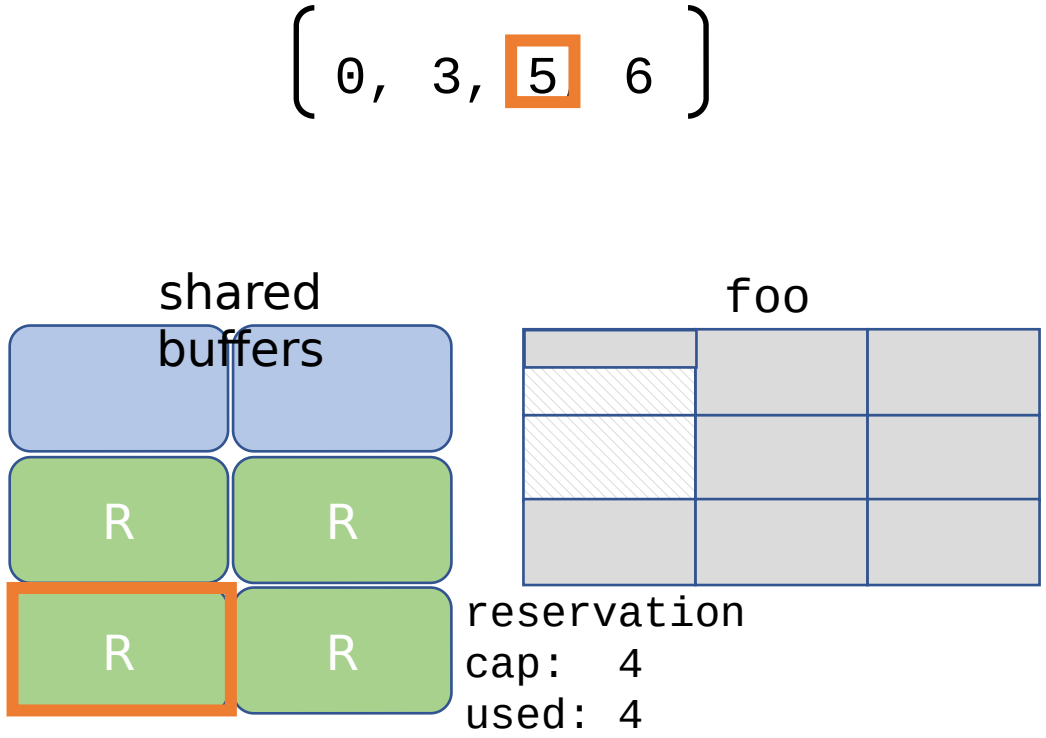
Postgres Autovacuum Workflow

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.



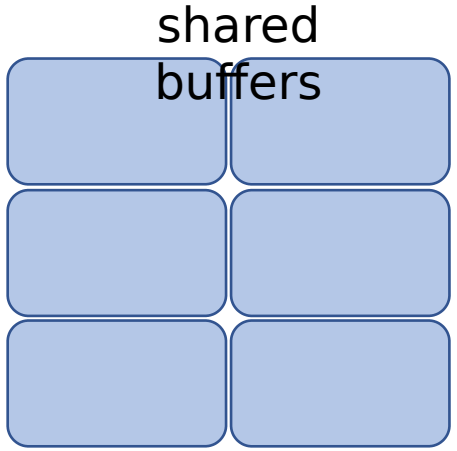
Postgres Autovacuum Workflow

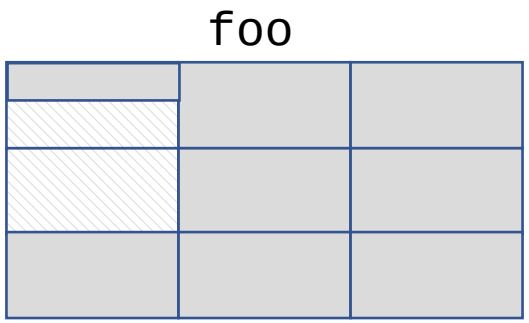
1. Identify the next block to vacuum.
2. Check for the block in shared buffers.
   i. If it is, vacuum it!
3. Find the next reserved buffer to use.
   i. If we are not at the reservation cap, evict a shared buffer.
   ii. If we are reusing a dirty, reserved buffer, flush it.
4. Read the block into the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



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.
6. Upon completing vacuum cycle, return all reserved buffers.

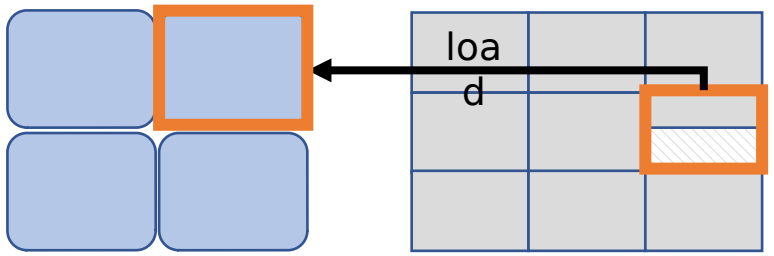




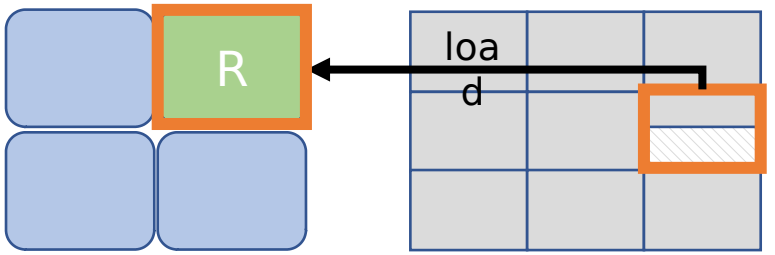
Why Track I/O Per Backend Type?

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

client backend read

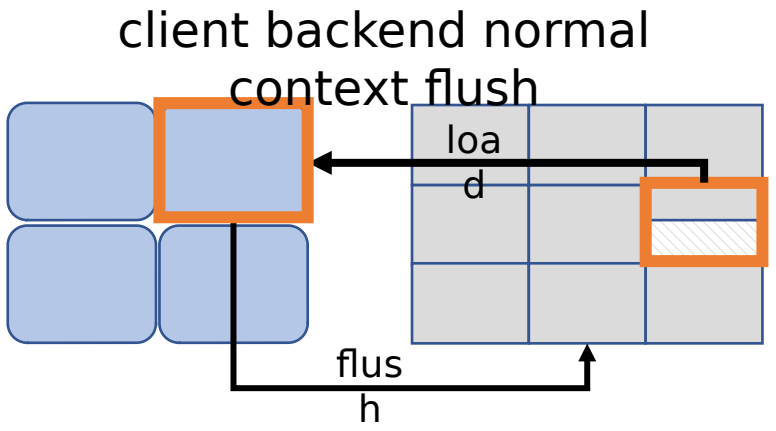


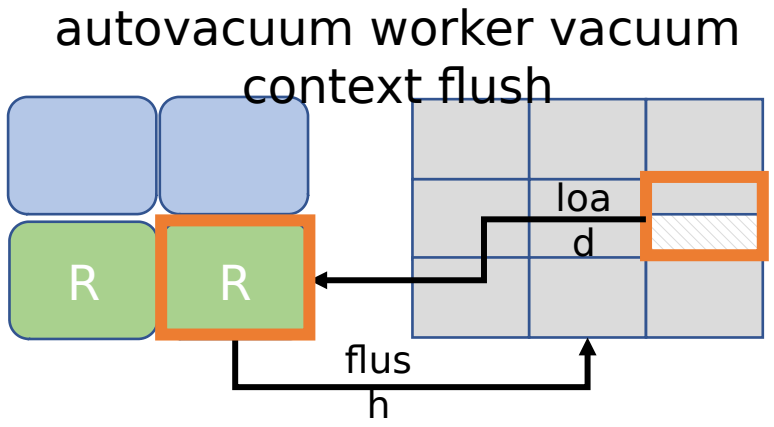
autovacuum worker read



Why Track I/O Per Context?

- Shared buffers not used for all I/O
- Vacuum I/O not in shared buffers

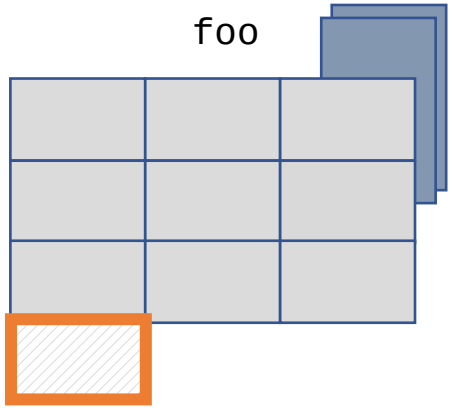




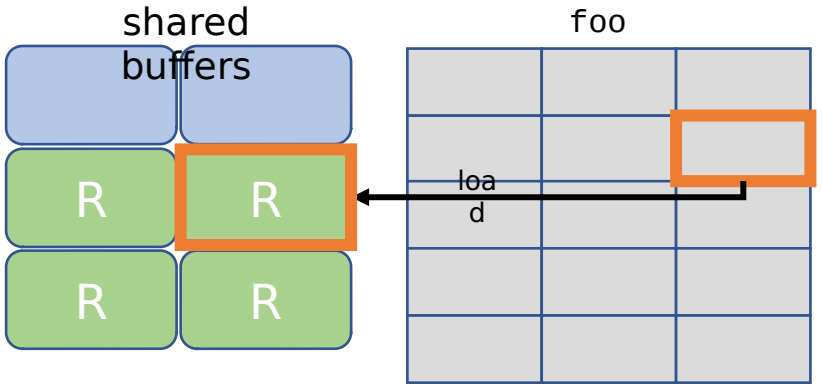
Analytic Workload I/O Characteristics

High number of extends during bulk load operations like

COPY FROM.

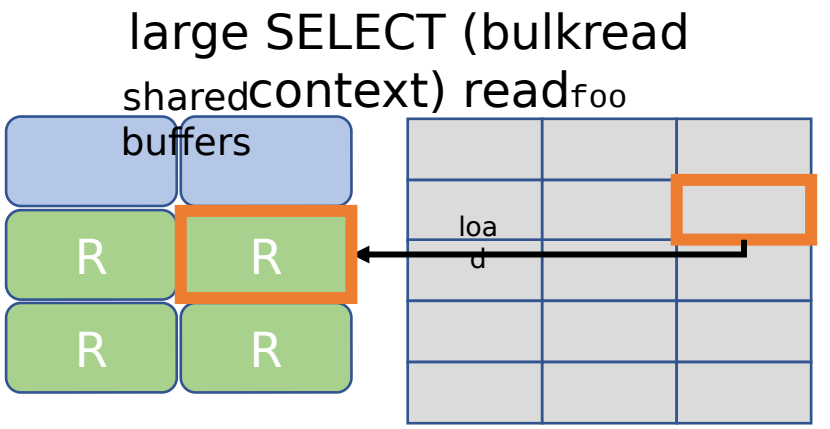


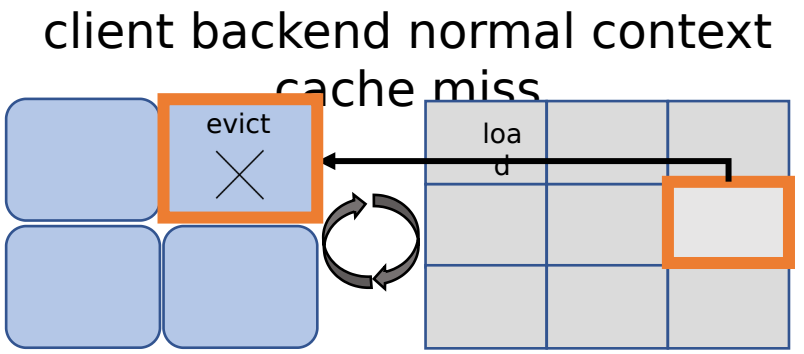
High number of reads during bulk read operations of data not in shared buffers.



Why Track I/O Per Context?

- Shared buffers not used for all I/O
- Large* SELECTs not in shared buffers

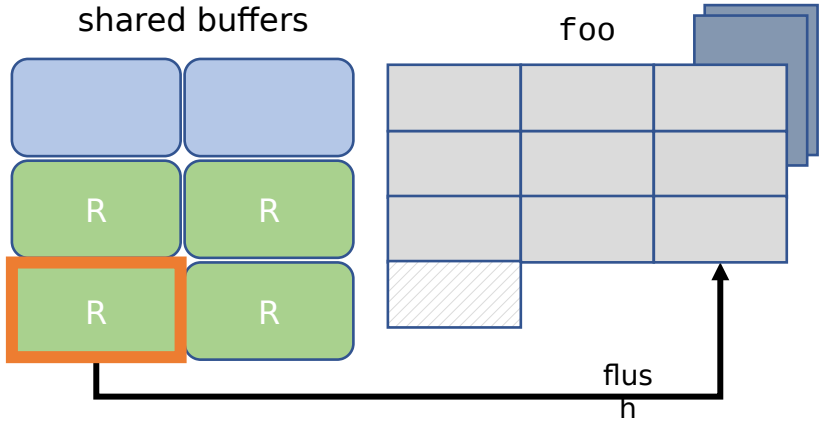




*large = table blocks > shared buffers / 4

Why Count Flushes and Extends Separately?

- COPY FROM does lots of extends
- Extends are normal for bulk writes



Data-Driven Tuning with pg_stat_io

Shared Buffers Too Small

```
  backend_type   | io_object | io_context | reads
-----------------+-----------+------------+-----------
 client backend | relation  | normal     | 128443922
```

- client_backend normal context reads high

Background Writer Too Passive

backend_type	io_object	io_context	writes
client backend	relation	normal	9986222
background writer	relation	normal	776549

- client backend normal context writes high
- background writer normal context writes high

Shared Buffers Not Too Small

```
  backend_type  | io_object | io_context | reads
----------------+-----------+------------+----------
 client backend | relation  | bulkread   |   9986222
 client backend | relation  | normal     |   210
```

- client backend normal context reads not high
- client backend bulkread context reads high

OR

- autovacuum worker vacuum context

Future additions

- I/O timing
- ”bypass” IO

Contact me:
@melanieplageman