Fast-path locking improvements in PG18

Tomas Vondra <vondratomas@microsoft.com> / <tomas@vondra.me>

pgconf.dev 2025, May 13-16, Montreal

Microsoft

Agenda

- Why this improvement?
- A bit of history (PG 9.2)
- PG 18 improvements
- Trade-offs
- Challenges
- Future

Why was I looking into this?

- end of 2023 (?)
- customer reports poor performance
- partitioned table (handful of partitions)
- upgraded from Xeon to EPYC
- expected better performance from EPYC
  - cores a bit "slower" but ~2x the core count
- the opposite happened (with concurrency)

example workload

- `pgbench -i -s 1 --partitions 10`

- `ALTER TABLE pgbench_accounts ADD COLUMN aid_parent INT;`
- `UPDATE pgbench_accounts SET aid_parent = aid;`
- `CREATE INDEX ON pgbench_accounts(aid_parent);`
- `VACUUM FULL pgbench_accounts;`

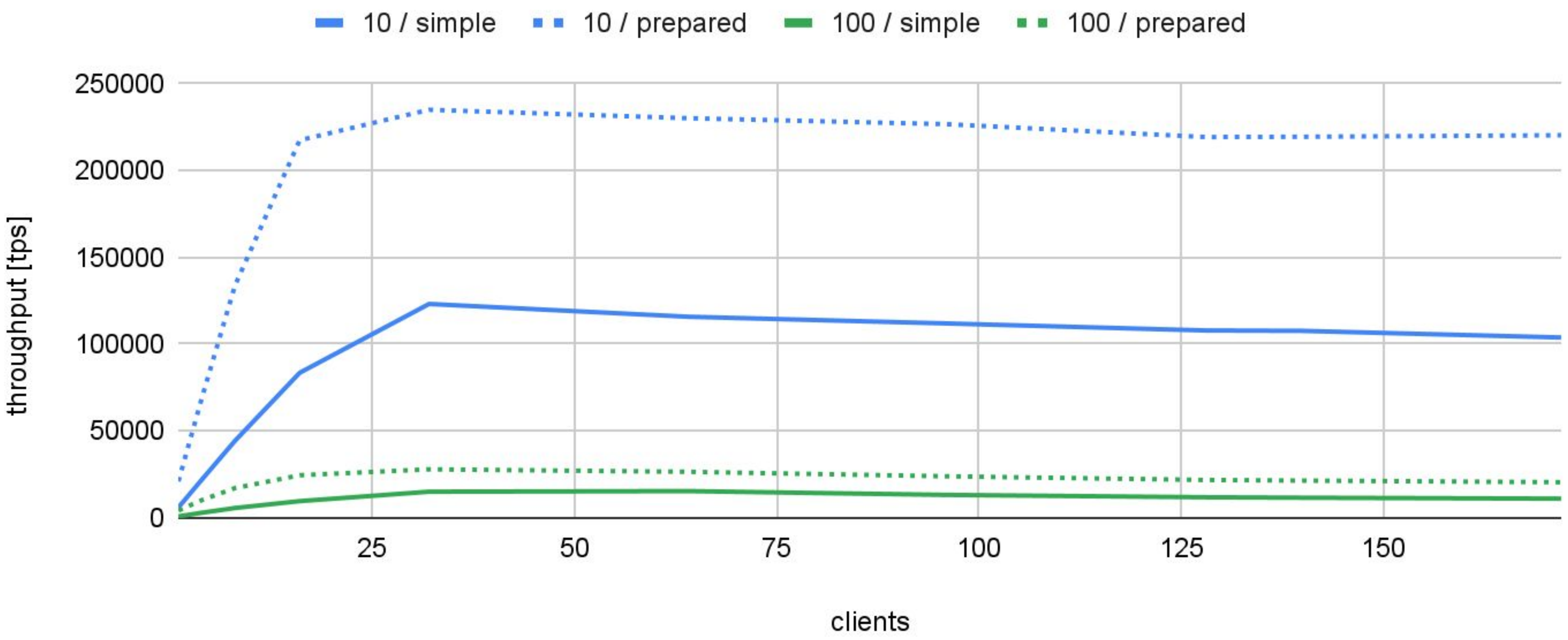
```
\set aid random(1, 100000 * :scale)
SELECT * FROM pgbench_accounts pa
         JOIN pgbench_branches pb ON (pa.bid = pb.bid)
 WHERE pa.aid_parent = :aid
```

EXPLAIN

```
                                                              QUERY PLAN
-----------------------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=1.52..34.41 rows=10 width=465)
   Hash Cond: (pa.bid = pb.bid)
   ->  Append  (cost=0.29..33.15 rows=10 width=101)
       ->  Index Scan using pgbench_accounts_1_aid_parent_idx on pgbench_accounts_1 pa_1 (cost=0.29..3.31 rows=1 width=101)
             Index Cond: (aid_parent = 3489734)
       ->  Index Scan using pgbench_accounts_2_aid_parent_idx on pgbench_accounts_2 pa_2 (cost=0.29..3.31 rows=1 width=101)
             Index Cond: (aid_parent = 3489734)
       ->  Index Scan using pgbench_accounts_3_aid_parent_idx on pgbench_accounts_3 pa_3 (cost=0.29..3.31 rows=1 width=101)
             Index Cond: (aid_parent = 3489734)
       ->  Index Scan using pgbench_accounts_4_aid_parent_idx on pgbench_accounts_4 pa_4 (cost=0.29..3.31 rows=1 width=101)
             Index Cond: (aid_parent = 3489734)
       -> ...
   ->  Hash  (cost=1.10..1.10 rows=10 width=364)
       ->  Seq Scan on pgbench_branches pb  (cost=0.00..1.10 rows=10 width=364)
```

throughput with partitions

AMD EPYC 9V74 80-Core Processor



What could be causing this?

- Clearly a concurrency issue.
- Something is contended, but what?
- Let's jump to "obvious" conclusions!

```
/* lwlock.h */
#define LOG2_NUM_LOCK_PARTITIONS  4
#define NUM_LOCK_PARTITIONS  (1 << LOG2_NUM_LOCK_PARTITIONS)
```

16

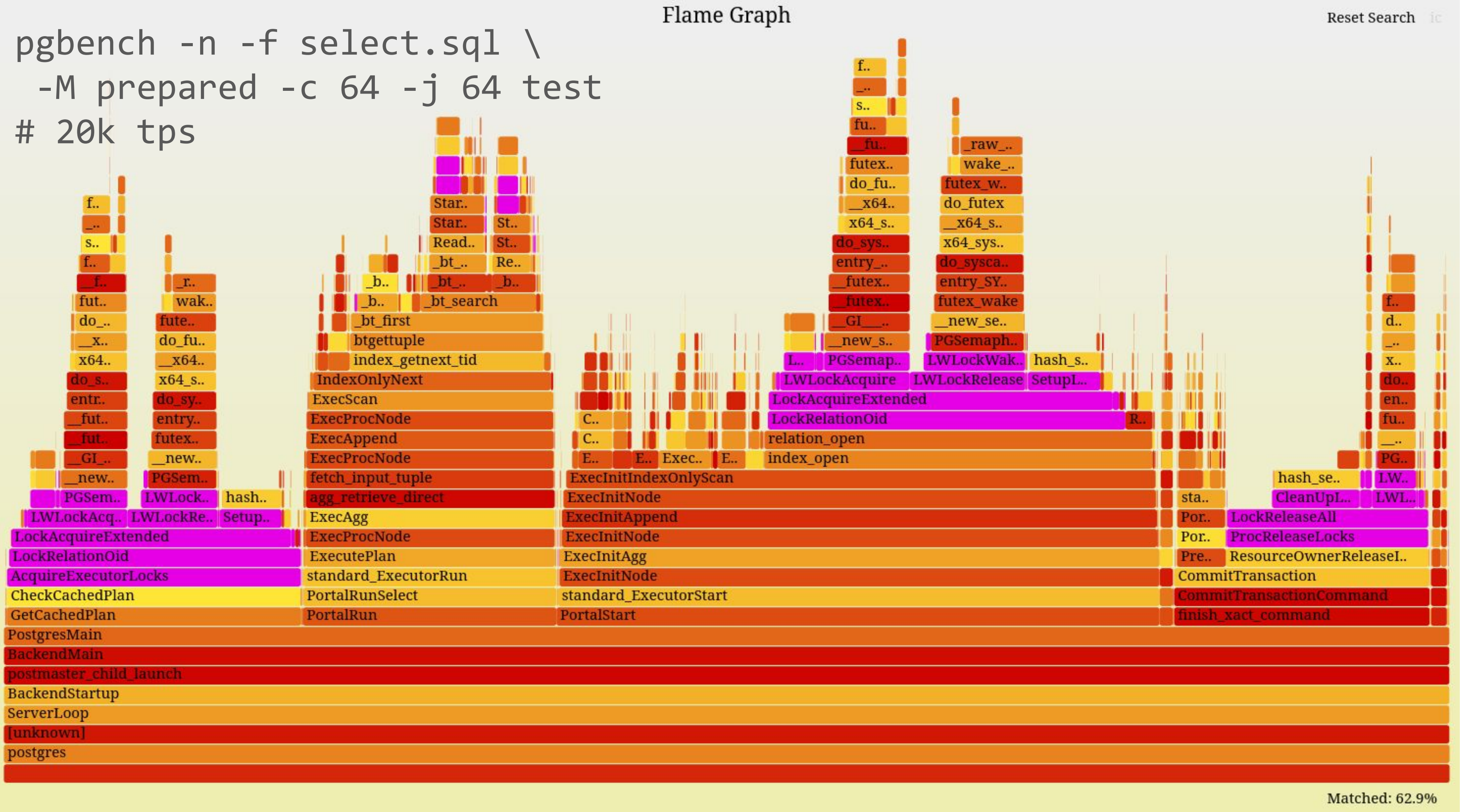
- This is not it. Increasing to 64 makes no difference.

Time for crazy ideas ...

- Could be power management / thermal throttling?
  - seen that before, was "fun" to investigate (invisible from a VM)
- Worse with SMT / hyper threading.
  - kinda sad to run with cores disabled
- Could it be malloc contention?
  - more about this later ...

```
Samples: 20K of event 'task-clock:ppp', Event count (approx.): 5181500000
  Children      Self  Command   Shared Object      Symbol
+   99.99%     0.00%  postgres  [unknown]          [.] 0xffffffffffffff
+   99.98%     0.00%  postgres  postgres           [.] ServerLoop
+   99.98%     0.00%  postgres  postgres           [.] BackendStartup (inlined)
+   99.98%     0.00%  postgres  postgres           [.] postmaster_child_launch
+   99.98%     0.00%  postgres  postgres           [.] BackendMain
+   99.98%     0.09%  postgres  postgres           [.] PostgresMain
+   44.91%     0.28%  postgres  postgres           [.] LockRelationOid
+   43.17%     1.45%  postgres  postgres           [.] LockAcquireExtended
+   41.73%     0.03%  postgres  postgres           [.] PortalStart
+   41.60%     0.03%  postgres  postgres           [.] standard_ExecutorStart
+   41.50%     0.22%  postgres  postgres           [.] ExecInitNode
+   41.49%     0.06%  postgres  postgres           [.] ExecInitAgg
+   41.16%     0.11%  postgres  postgres           [.] ExecInitAppend
+   40.77%     0.41%  postgres  postgres           [.] ExecInitIndexOnlyScan
+   28.50%     0.55%  postgres  postgres           [.] relation_open
+   26.86%     0.01%  postgres  postgres           [.] index_open
+   24.69%     0.00%  postgres  [kernel.kallsyms]  [k] entry_SYSCALL_64_after_hwframe
+   24.58%     1.26%  postgres  [kernel.kallsyms]  [k] do_syscall_64
+   22.42%     0.10%  postgres  [kernel.kallsyms]  [k] x64_sys_call
+   20.67%     0.12%  postgres  [kernel.kallsyms]  [k] __x64_sys_futex
+   20.52%     0.04%  postgres  [kernel.kallsyms]  [k] do_futex
+   20.36%     0.01%  postgres  postgres           [.] GetCachedPlan
+   20.31%     0.00%  postgres  postgres           [.] CheckCachedPlan (inlined)
+   20.31%     0.14%  postgres  postgres           [.] AcquireExecutorLocks
+   20.07%     8.12%  postgres  postgres           [.] LWLockAcquire
+   18.25%    16.02%  postgres  postgres           [.] hash_search_with_hash_value
+   17.68%     3.82%  postgres  postgres           [.] LWLockRelease
+   17.67%     0.00%  postgres  postgres           [.] finish_xact_command (inlined)
+   17.67%     0.01%  postgres  postgres           [.] CommitTransactionCommand
+   17.66%     0.05%  postgres  postgres           [.] CommitTransaction
+   17.52%     0.00%  postgres  postgres           [.] PortalRun
```

```
pgbench -n -f select.sql \
 -M prepared -c 64 -j 64 test
# 20k tps
```



Locking relations

- backend locking a relation (OID)
- shared lock table (LOCK/PROCLOCK)
- partitioned but expensive to update

Fast-path locking (9.2)

Table 13.2. Conflicting Lock Modes

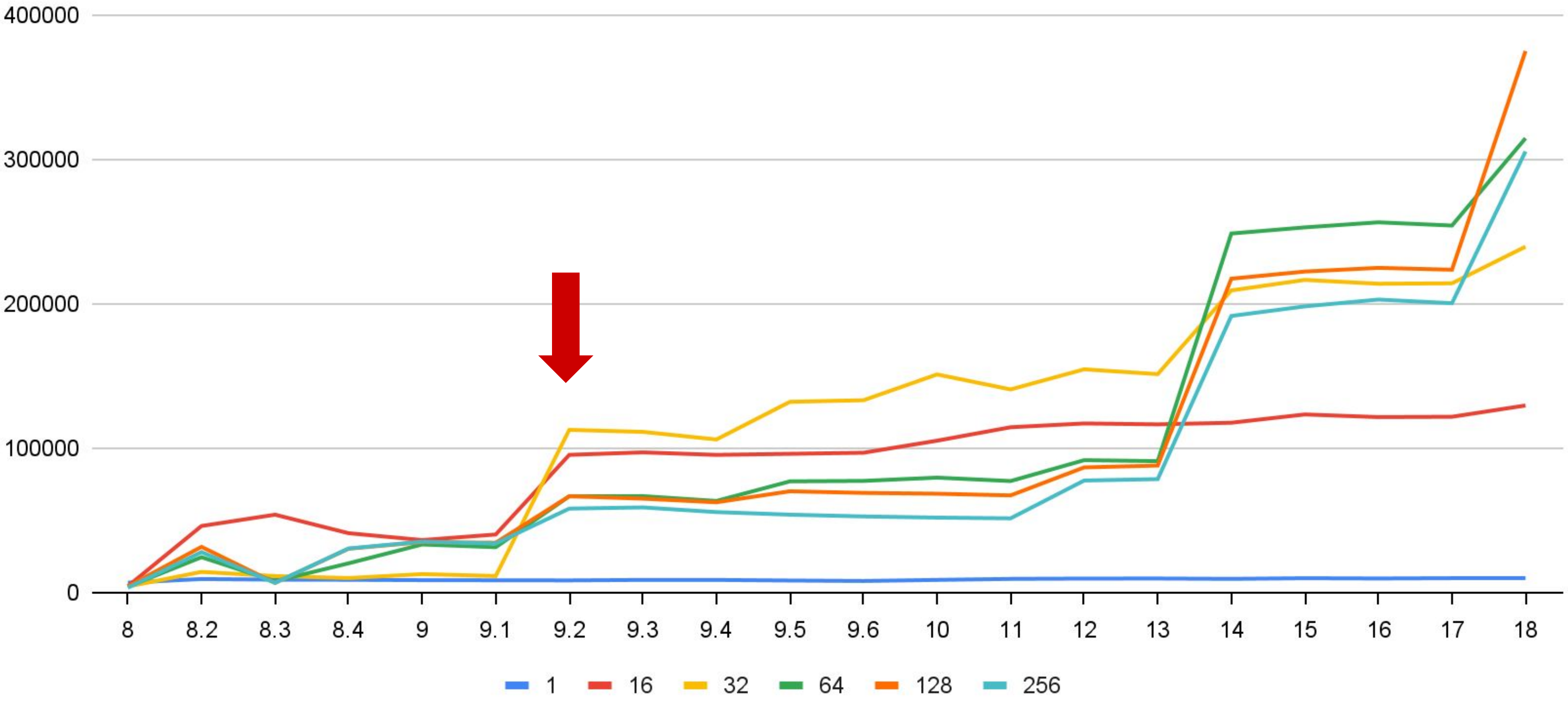
Requested Lock Mode	Existing Lock Mode							
	ACCESS SHARE	ROW SHARE	ROW EXCL.	SHARE UPDATE EXCL.	SHARE	SHARE ROW EXCL.	EXCL.	ACCESS EXCL.
ACCESS SHARE								X
ROW SHARE							X	X
ROW EXCL.					X	X	X	X
SHARE UPDATE EXCL.				X	X	X	X	X
SHARE			X	X		X	X	X
SHARE ROW EXCL.			X	X	X	X	X	X
EXCL.		X	X	X	X	X	X	X
ACCESS EXCL.	X	X	X	X	X	X	X	X

https://www.postgresql.org/docs/current/explicit-locking.html

Fast-path locking (9.2)

- fast-path array in PGPROC
  - "local cache" - the point is to not use shared hash table often
  - still in shared memory, but has a separate lock (per process)
- fast-path protocol (lock.c, LockAcquireExtended)
  - fast-path if no one holds a conflicting lock + there's space in PGPROC
  - obtaining conflicting lock -> transfer locks to shared hash table
- capacity for 16 OIDs - that's not very many
  - tables + indexes + ...
  - trivial to hit the limit, especially with partitioning

OLTP starjoin / -M prepared



https://vondra.me/pdf/performance-archaeology-pgconfeu-2024.pdf

Making it larger ...

- can't keep it in PGPROC anymore
  - still has to be shared memory
  - but allocated as a separate "chunk"
- also, make it configurable
  - so that people can adjust that by a GUC
- no change to the fast-path locking protocol
- But what should be the data structure?

Data structure

Microsoft

- array + linear search
  - worked great for 16 items, linear search wins here
  - probably not beyond 32/64 items, we're aiming for 1024+
- hash table (open addressing)
  - we'd need to limit load factor (e.g. 75%) to keep it fast
  - random access is not great (cacheline 64B)
- 16-way set-associative cache
  - hash table of arrays
  - ingenious product of my laziness

16-way set-associative cache

https://en.algorithmica.org/hpc/cpu-cache/associativity/

https://developer.arm.com/documentation/den0013/d/Caches/Cache-architecture/Set-associative-caches

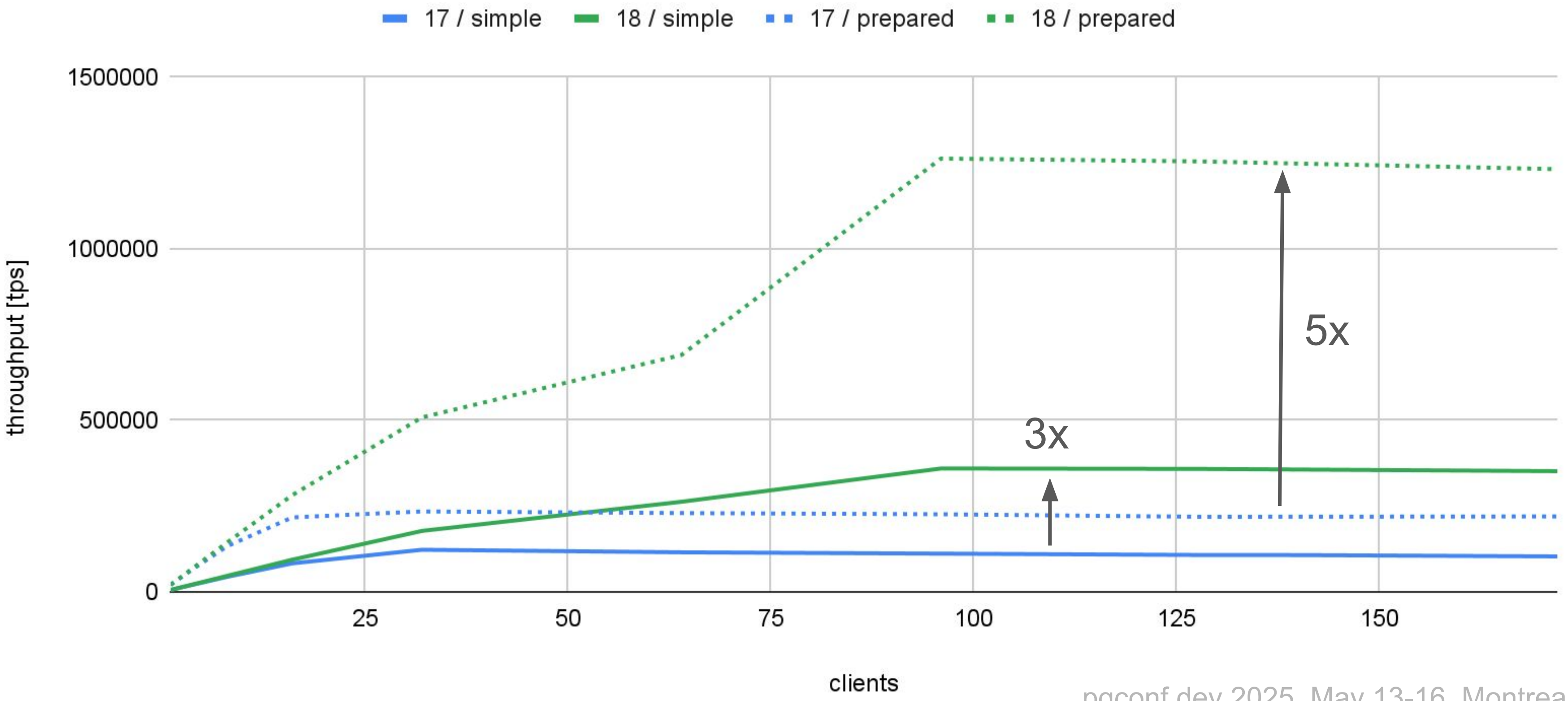
16-way set-associative cache

- simple concept
  - hash + array
- nice sequential access
  - regular hash tables are much more random
  - not great, even for RAM
  - cache friendly (cachelines)
- no problem with limited capacity
  - can always promote to shared lock table

10 partitions, max_locks_per_transaction = 64

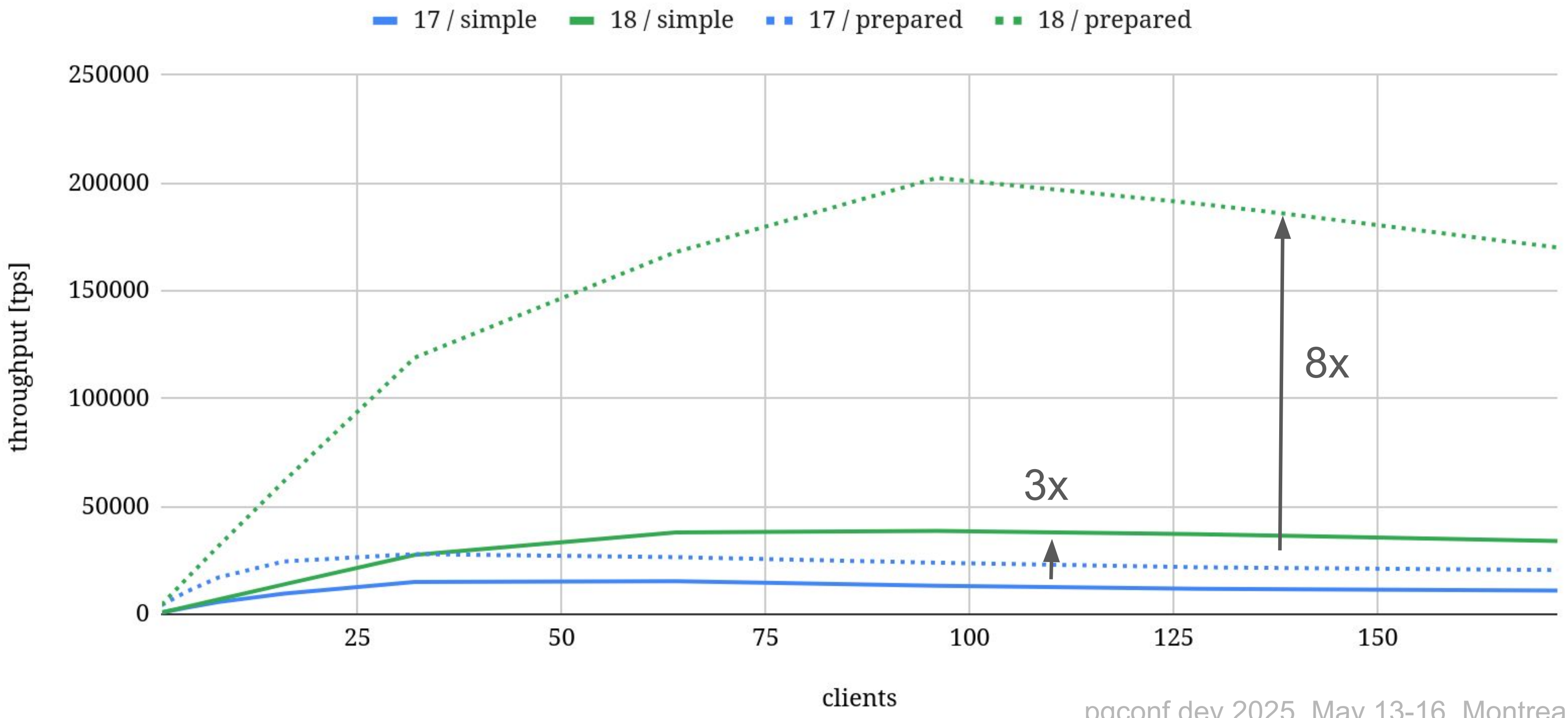
AMD EPYC 9V74 80-Core Processor

Microsoft



100 partitions, max_locks_per_transaction = 1024

AMD EPYC 9V74 80-Core Processor



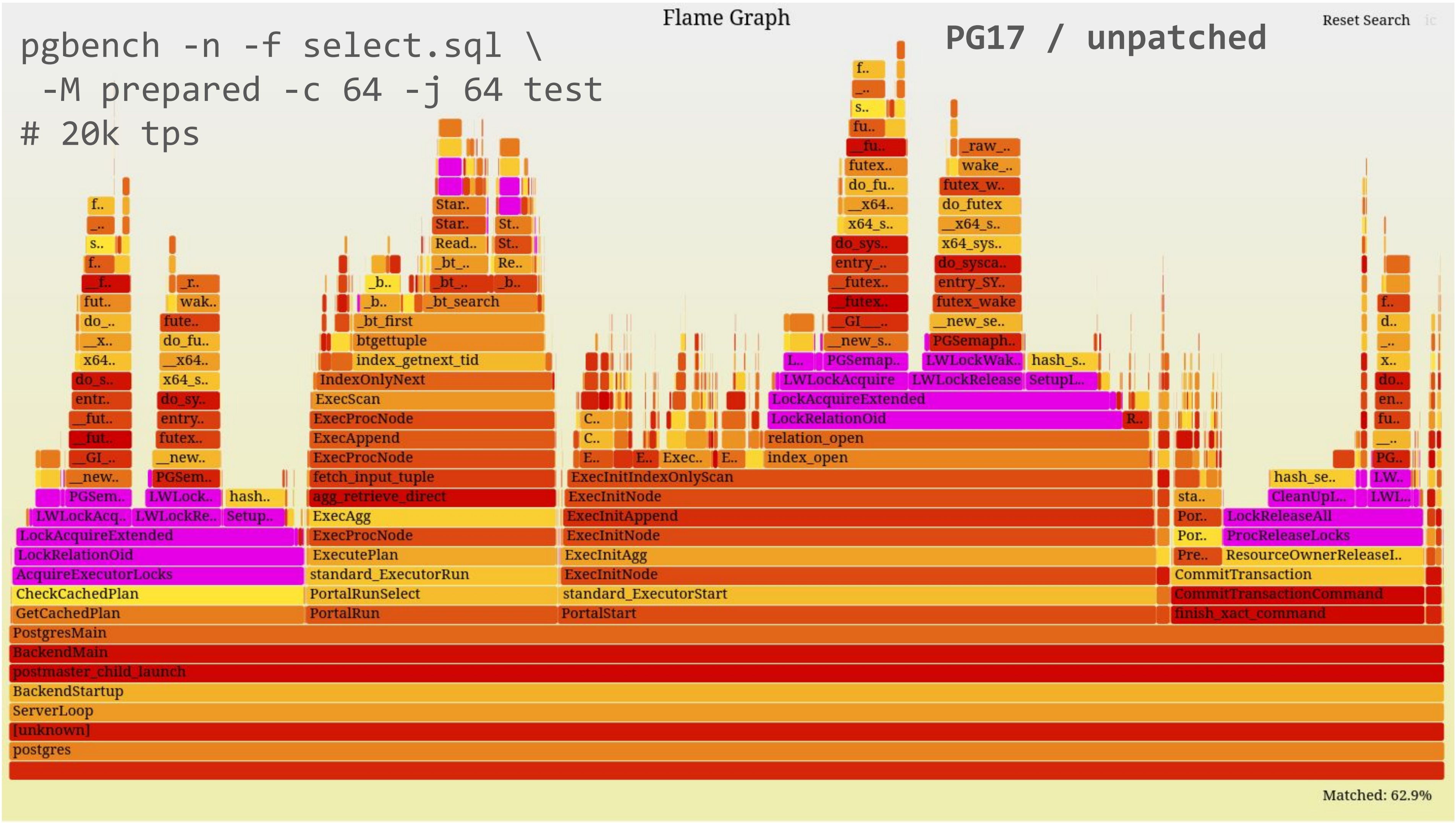
Trade-offs

- tied to max_locks_per_transaction
  - ease of tuning vs. configurability (too many GUCs)
  - best idea about how many locks to expect
  - per-backend limit (max_locks_per_transaction was not that)
- what's a good value?
  - no "optimal" value, depends on workload
  - fast-path locks are cheaper (smaller) than shared lock table entries
- max_locks_per_transaction = 64
  - sensible, maybe not ideal for "unbalanced" clusters?
  - should be enough for ~10 tables

Challenges

benchmark

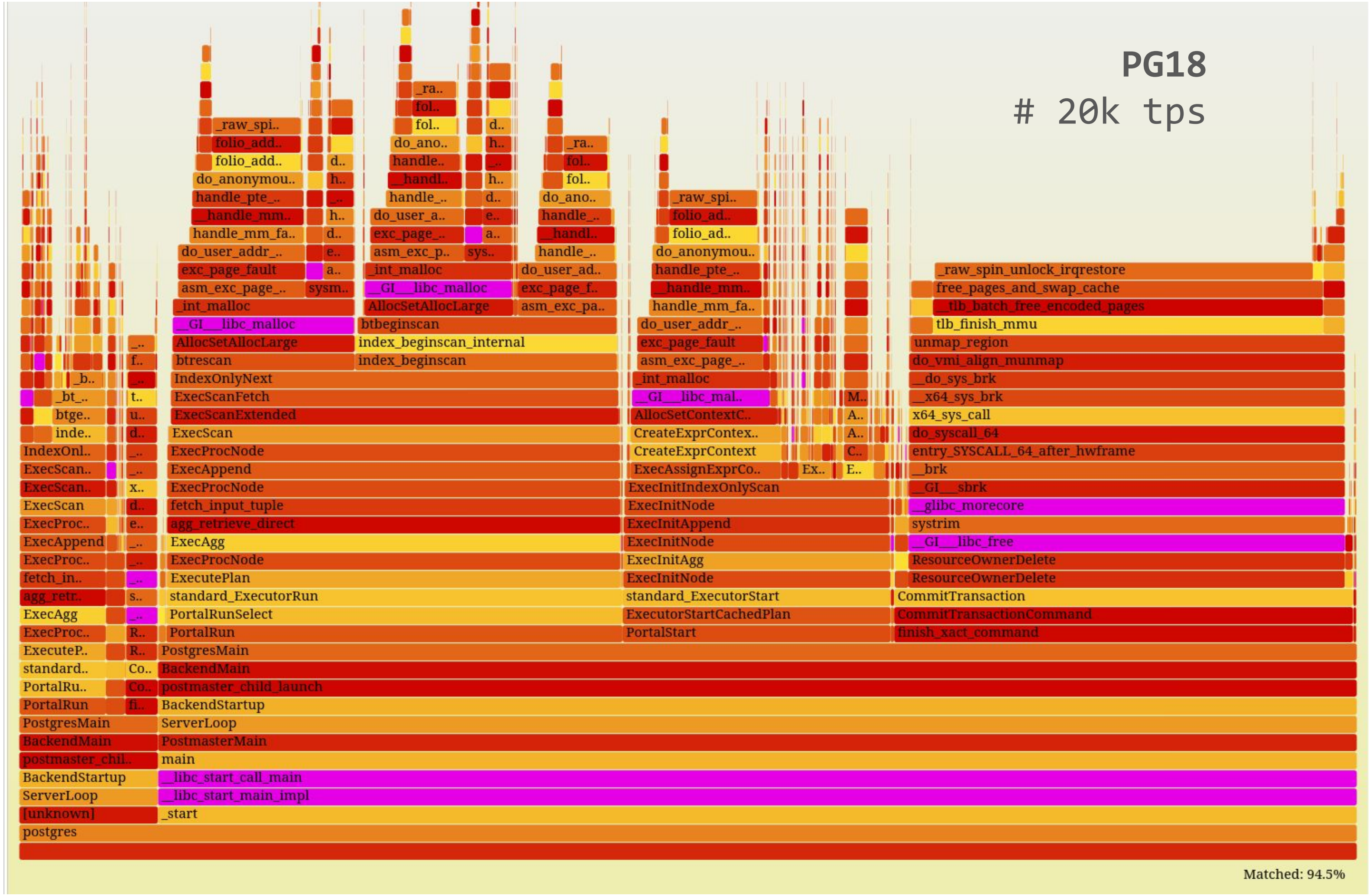
```
pgbench -i -s 1 --partitions 100 test


update pgbench_accounts set bid = aid;

create index on pgbench_accounts (bid);


# select.sql

select count(*) from pggench_accounts where bid = 0;
```


pgbench -n -f select.sql \
-M prepared -c 64 -j 64 test
20k tps
Flame Graph
PG17 / unpatched
Reset Search
postgres
[unknown]
ServerLoop
BackendStartup
postmaster_child_launch
BackendMain
PostgresMain
GetCachedPlan
CheckCachedPlan
AcquireExecutorLocks
LockRelationOid
LockAcquireExtended
LWLockAcq..
LWLockRe..
Setup..
PGSem..
LWLock..
hash..
PortalRun
PortalRunSelect
standard_ExecutorRun
ExecutePlan
ExecProcNode
ExecAgg
agg_retrieve_direct
fetch_input_tuple
ExecProcNode
ExecAppend
ExecProcNode
ExecScan
IndexOnlyNext
index_getnext_tid
btgettuple
_bt_first
_bt_search
PortalStart
standard_ExecutorStart
ExecInitNode
ExecInitAgg
ExecInitNode
ExecInitAppend
ExecInitNode
ExecInitIndexOnlyScan
index_open
relation_open
LockRelationOid
LockAcquireExtended
LWLockAcquire
LWLockRelease
SetupL..
PGSemap..
LWLockWak..
hash_s..
PGSemaph..
futex_wake
futex_w..
do_futex
finish_xact_command
CommitTransactionCommand
CommitTransaction
ResourceOwnerReleaseI..
ProcReleaseLocks
LockReleaseAll
CleanUpL..
hash_se..
Matched: 62.9%

```
Samples: 20K of event 'task-clock:ppp', Event count (approx.): 5181500000
  Children      Self  Command   Shared Object      Symbol
+   99.99%     0.00%  postgres  [unknown]          [.] 0xffffffffffffff
+   99.98%     0.00%  postgres  postgres           [.] ServerLoop
+   99.98%     0.00%  postgres  postgres           [.] BackendStartup (inlined)
+   99.98%     0.00%  postgres  postgres           [.] postmaster_child_launch
+   99.98%     0.00%  postgres  postgres           [.] BackendMain
+   99.98%     0.09%  postgres  postgres           [.] PostgresMain
+   44.91%     0.28%  postgres  postgres           [.] LockRelationOid
+   43.17%     1.45%  postgres  postgres           [.] LockAcquireExtended
+   41.73%     0.03%  postgres  postgres           [.] PortalStart
+   41.60%     0.03%  postgres  postgres           [.] standard_ExecutorStart
+   41.50%     0.22%  postgres  postgres           [.] ExecInitNode
+   41.49%     0.06%  postgres  postgres           [.] ExecInitAgg
+   41.16%     0.11%  postgres  postgres           [.] ExecInitAppend
+   40.77%     0.41%  postgres  postgres           [.] ExecInitIndexOnlyScan
+   28.50%     0.55%  postgres  postgres           [.] relation_open
+   26.86%     0.01%  postgres  postgres           [.] index_open
+   24.69%     0.00%  postgres  [kernel.kallsyms]  [k] entry_SYSCALL_64_after_hwframe
+   24.58%     1.26%  postgres  [kernel.kallsyms]  [k] do_syscall_64
+   22.42%     0.10%  postgres  [kernel.kallsyms]  [k] x64_sys_call
+   20.67%     0.12%  postgres  [kernel.kallsyms]  [k] __x64_sys_futex
+   20.52%     0.04%  postgres  [kernel.kallsyms]  [k] do_futex
+   20.36%     0.01%  postgres  postgres           [.] GetCachedPlan
+   20.31%     0.00%  postgres  postgres           [.] CheckCachedPlan (inlined)
+   20.31%     0.14%  postgres  postgres           [.] AcquireExecutorLocks
+   20.07%     8.12%  postgres  postgres           [.] LWLockAcquire
+   18.25%    16.02%  postgres  postgres           [.] hash_search_with_hash_value
+   17.68%     3.82%  postgres  postgres           [.] LWLockRelease
+   17.67%     0.00%  postgres  postgres           [.] finish_xact_command (inlined)
+   17.67%     0.01%  postgres  postgres           [.] CommitTransactionCommand
+   17.66%     0.05%  postgres  postgres           [.] CommitTransaction
+   17.52%     0.00%  postgres  postgres           [.] PortalRun
```

PG18

20k tps

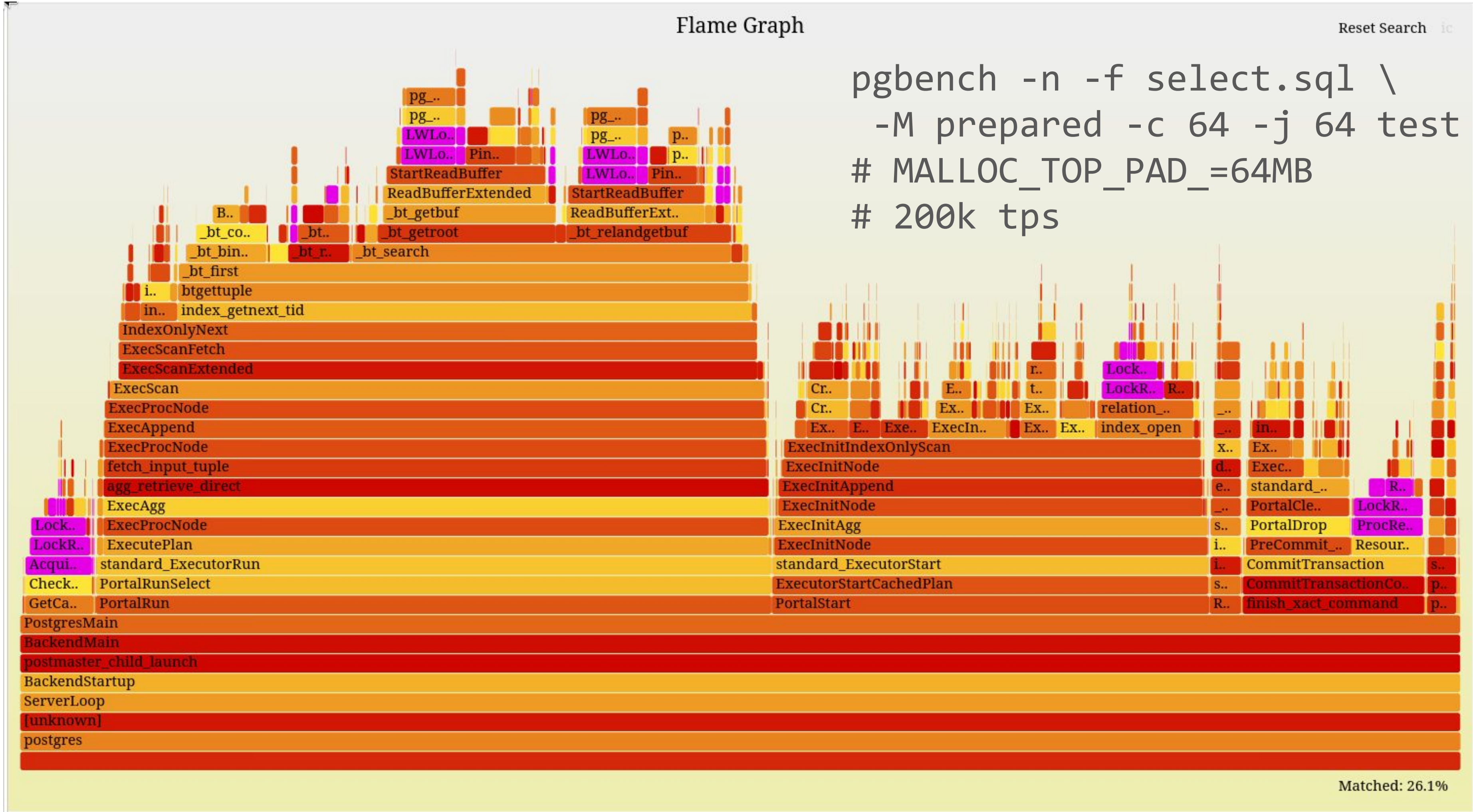


postgres
[unknown]
ServerLoop
BackendStartup
postmaster_chil..
BackendMain
PostgresMain
PortalRun
PortalRu..
standard..
ExecuteP..
ExecProc..
ExecAgg
agg_retr..
fetch_in..
ExecProc..
ExecAppend
ExecProc..
ExecScan
ExecScan..
ExecScan..
IndexOnl..
inde..
btge..
_bt_..
_b..
_start
__libc_start_main_impl
__libc_start_call_main
main
PostmasterMain
ServerLoop
BackendStartup
postmaster_child_launch
BackendMain
PostgresMain
PortalRun
PortalRunSelect
standard_ExecutorRun
ExecutePlan
ExecProcNode
ExecAgg
agg_retrieve_direct
fetch_input_tuple
ExecProcNode
ExecAppend
ExecProcNode
ExecScan
ExecScanExtended
ExecScanFetch
IndexOnlyNext
btrescan
AllocSetAllocLarge
__GI___libc_malloc
_int_malloc
asm_exc_page_..
exc_page_fault
do_user_addr_..
handle_mm_fa..
__handle_mm..
handle_pte_..
do_anonymou..
folio_add..
folio_add..
_raw_spi..
index_beginscan
index_beginscan_internal
btbeginscan
AllocSetAllocLarge
__GI___libc_malloc
_int_malloc
asm_exc_p..
exc_page_..
do_user_a..
handle_..
__handl..
handle..
do_ano..
fol..
fol..
_ra..
asm_exc_pa..
exc_page_f..
do_user_ad..
handle_..
__handl..
handle_..
do_ano..
fol..
fol..
_ra..
PortalStart
ExecutorStartCachedPlan
standard_ExecutorStart
ExecInitNode
ExecInitAgg
ExecInitNode
ExecInitAppend
ExecInitNode
ExecInitIndexOnlyScan
ExecAssignExprCo..
CreateExprContext
CreateExprContex..
AllocSetContextC..
__GI___libc_mal..
_int_malloc
asm_exc_page_..
exc_page_fault
do_user_addr_..
handle_mm_fa..
__handle_mm..
handle_pte_..
do_anonymou..
folio_ad..
folio_ad..
_raw_spi..
finish_xact_command
CommitTransactionCommand
CommitTransaction
ResourceOwnerDelete
ResourceOwnerDelete
__GI___libc_free
systrim
__glibc_morecore
__GI___sbrk
__brk
entry_SYSCALL_64_after_hwframe
do_syscall_64
x64_sys_call
__x64_sys_brk
__do_sys_brk
do_vmi_align_munmap
unmap_region
tlb_finish_mmu
__tlb_batch_free_encoded_pages
free_pages_and_swap_cache
_raw_spin_unlock_irqrestore
Matched: 94.5%

```
Samples: 238K of event 'task-clock:ppp', Event count (approx.): 59503000000
  Children      Self  Command   Shared Object        Symbol
+   99.99%     0.00%  postgres  postgres             [.] ServerLoop
+   99.99%     0.00%  postgres  postgres             [.] BackendStartup (inlined)
+   99.99%     0.00%  postgres  postgres             [.] postmaster_child_launch
+   99.99%     0.00%  postgres  postgres             [.] BackendMain
+   99.99%     0.03%  postgres  postgres             [.] PostgresMain
+   89.63%     0.00%  postgres  postgres             [.] _start
+   89.63%     0.00%  postgres  libc.so.6            [.] __libc_start_main_impl (inlined)
+   89.63%     0.00%  postgres  libc.so.6            [.] __libc_start_call_main
+   89.63%     0.00%  postgres  postgres             [.] main
+   89.63%     0.00%  postgres  postgres             [.] PostmasterMain
+   58.43%    58.33%  postgres  [kernel.kallsyms]    [k] _raw_spin_unlock_irqrestore
+   47.26%     0.00%  postgres  [kernel.kallsyms]    [k] asm_exc_page_fault
+   47.25%     0.01%  postgres  [kernel.kallsyms]    [k] exc_page_fault
+   47.13%     6.47%  postgres  [kernel.kallsyms]    [k] do_user_addr_fault
+   40.56%     0.00%  postgres  postgres             [.] PortalRun
+   40.56%     0.00%  postgres  postgres             [.] PortalRunSelect
+   40.55%     0.00%  postgres  postgres             [.] standard_ExecutorRun
+   40.32%     0.00%  postgres  postgres             [.] ExecutePlan (inlined)
+   40.32%     0.00%  postgres  postgres             [.] ExecProcNode (inlined)
+   40.32%     0.01%  postgres  postgres             [.] ExecAgg
+   40.32%     0.00%  postgres  postgres             [.] agg_retrieve_direct (inlined)
+   40.29%     0.00%  postgres  postgres             [.] fetch_input_tuple
+   40.29%     0.00%  postgres  postgres             [.] ExecProcNode (inlined)
+   40.29%     0.08%  postgres  postgres             [.] ExecAppend
+   40.25%     0.00%  postgres  postgres             [.] ExecProcNode (inlined)
+   40.14%     0.17%  postgres  postgres             [.] ExecScan
+   39.88%     0.86%  postgres  [kernel.kallsyms]    [k] handle_mm_fault
+   39.85%     0.00%  postgres  postgres             [.] ExecScanExtended (inlined)
+   39.85%     0.00%  postgres  postgres             [.] ExecScanFetch (inlined)
+   39.85%     0.11%  postgres  postgres             [.] IndexOnlyNext
+   38.98%     0.56%  postgres  [kernel.kallsyms]    [k] __handle_mm_fault
+   38.63%     0.00%  postgres  [kernel.kallsyms]    [k] entry_SYSCALL_64_after_hwframe
```

Other bottlenecks

- glibc malloc vs. concurrency
  - btbeginscan() allocates ~30kB, can't be cached, always malloc
  - MALLOC_TOP_PAD_ (see mallopt)
  - two "connected" bottlenecks - have to address both
  - jemalloc/tmalloc do not have this issue
- join order planning
  - OLTP starjoin
  - other bottleneck swamping the results
- multiple bottlenecks can be hit simultaneously
  - and compose in non-linear way (50% vs. 10x speedup)


Flame Graph
Reset Search
pgbench -n -f select.sql \
-M prepared -c 64 -j 64 test
MALLOC_TOP_PAD_=64MB
200k tps
pg_..
LWLo..
Pin..
StartReadBuffer
ReadBufferExtended
_bt_getbuf
_bt_getroot
_bt_relandgetbuf
ReadBufferExt..
_bt_co..
_bt_bin..
_bt_search
_bt_first
btgettuple
index_getnext_tid
IndexOnlyNext
ExecScanFetch
ExecScanExtended
ExecScan
ExecProcNode
ExecAppend
fetch_input_tuple
agg_retrieve_direct
ExecAgg
ExecutePlan
standard_ExecutorRun
PortalRunSelect
PortalRun
Lock..
LockR..
Acqui..
Check..
GetCa..
ExecInitIndexOnlyScan
ExecInitNode
ExecInitAppend
ExecInitAgg
standard_ExecutorStart
ExecutorStartCachedPlan
PortalStart
index_open
relation_..
standard_..
PortalCle..
PortalDrop
PreCommit_..
CommitTransaction
CommitTransactionCo..
finish_xact_command
ProcRe..
Resour..
PostgresMain
BackendMain
postmaster_child_launch
BackendStartup
ServerLoop
[unknown]
postgres
Matched: 26.1%

Future

- could we use the same idea elsewhere?
  - pins for "hot" buffers - maybe a "fast-path pinning"?
  - Problem #4 - Buffer Lock Contention (https://youtu.be/V75KpACdl6E?t=2120)
- consider hotness
  - now first come, first served
  - Maybe consider how often an OID is locked? Has to be cheap.
- NUMA effects
  - maybe should be NUMA partitioned
  - same NUMA node as PGPROC?
- make shared lock table cheaper
  - smaller entries, ...

Shout-out

- Robert Haas
  - wrote the fast-path locking in 9.2
  - it was extremely easy to build on his code
  - first PoC patch in ~ ½ day, worked on 1st try

- Jakub Wartak
  - support engineer / hacker in EDB investigating this
  - provided a lot of great insights and expertise
  - super-fun collaboration

Tomas Vondra

Microsoft

- Postgres engineer @ Microsoft
- https://vondra.me
- vondratomas@microsoft.com
- tomas@vondra.me
- office hours
- ...