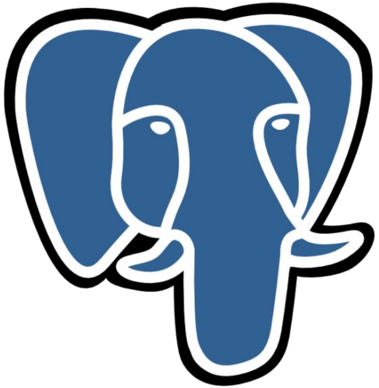
High Performance JSON PostgreSQL vs. MongoDB

FOSDEM PGDay 2018

Dominic Dwyer
Wei Shan Ang

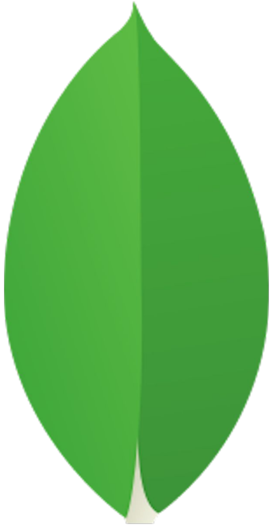
GlobalSign®
GMO INTERNET GROUP



PostgreSQL

VS

mongoDB®



GlobalSign

- GlobalSign identity & crypto services provider
- WebTrust certified Certificate Authority - 3rd in the world
- High volume services - IoT devices, cloud providers
- Cryptographic identities, timestamping, signing, etc

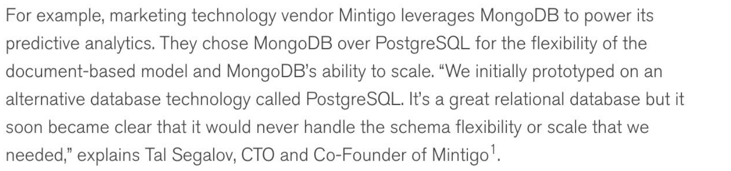
GlobalSign®
GMO INTERNET GROUP

About Me

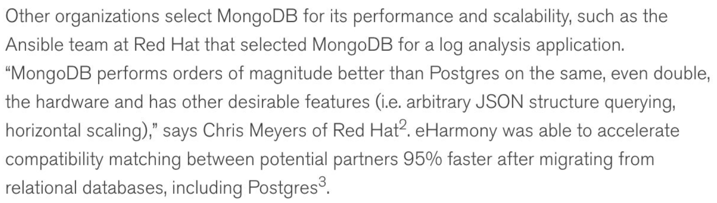
- Database Engineer with 6 years of experience
- From Singapore but now based in London
- Originally worked on Oracle systems
- Switched to open-source databases in 2015
- Currently working in GlobalSign as a “Data Reliability Engineer”

Motivation

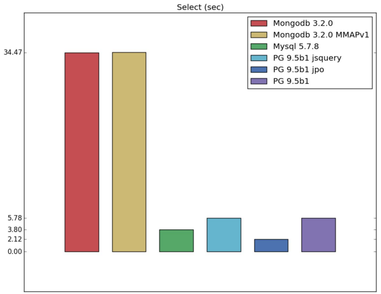
- Most benchmark results are biased - commercial interests
- We feel that benchmark results are measured “creatively”
- We use PostgreSQL and MongoDB a lot!
- We wanted to test the latest versions

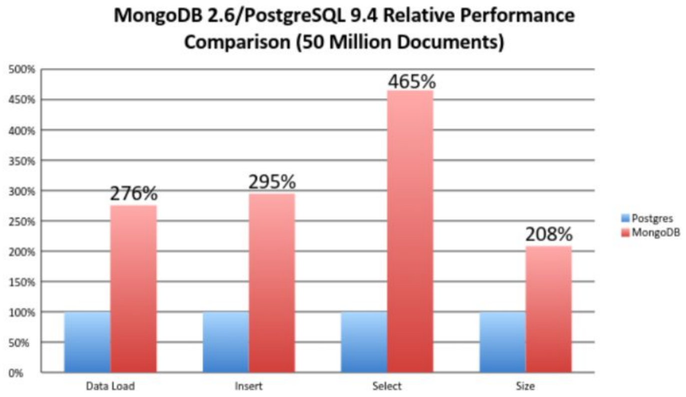


For example, marketing technology vendor Mintigo leverages MongoDB to power its predictive analytics. They chose MongoDB over PostgreSQL for the flexibility of the document-based model and MongoDB’s ability to scale. “We initially prototyped on an alternative database technology called PostgreSQL. It’s a great relational database but it soon became clear that it would never handle the schema flexibility or scale that we needed,” explains Tal Segalov, CTO and Co-Founder of Mintigo[1].



Other organizations select MongoDB for its performance and scalability, such as the Ansible team at Red Hat that selected MongoDB for a log analysis application. “MongoDB performs orders of magnitude better than Postgres on the same, even double, the hardware and has other desirable features (i.e. arbitrary JSON structure querying, horizontal scaling),” says Chris Meyers of Red Hat[2]. eHarmony was able to accelerate compatibility matching between potential partners 95% faster after migrating from relational databases, including Postgres[3].



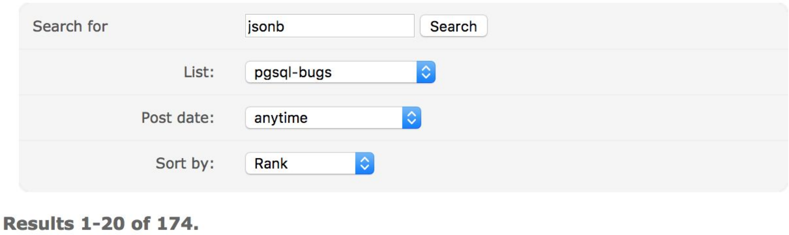


I see no reason to use Mongodb,

PostgreSQL still beats Mongodb !

PostgreSQL

PostgreSQL



Search for jsonb

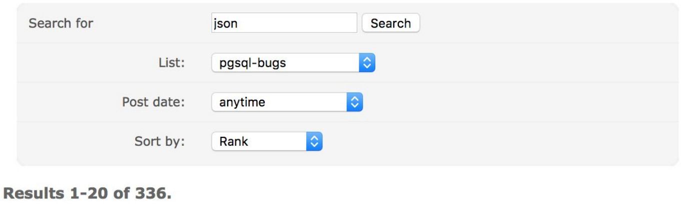
List: pgsql-bugs

Post date: anytime

Sort by: Rank

Results 1-20 of 174.

- Around for 21 years
- JSON supported since 9.2
- JSONB supported since 9.4
- Does not have *any* statistics about the internals of document types like JSON or JSONB
  - Can be overcome with default_statistics_target or ALTER TABLE TABLE_NAME ALTER int4 SET STATISTICS 2000;
- Many JSON/JSONB operator/functions released since 9.2 (jsonb_set, jsonb_insert)
- Many JSON/JSONB bug fixes too



Search for json

List: pgsql-bugs

Post date: anytime

Sort by: Rank

Results 1-20 of 336.

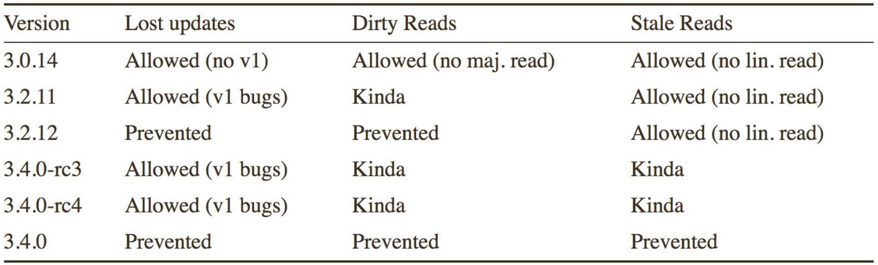
PostgreSQL Ecosystem

- “Build it yourself”
- Many High Availability solutions - all 3rd party
  - repmgr, pacemaker/corosync, Slony, Patroni and many more
- Connection Pooling
  - pgBouncer (single-threaded), pgpool-II
- Sharding
  - CitusDB
- Live version upgrades - tricky!
  - pg_upgrade, Slony, pg_dump and pg_logical

MongoDB

MongoDB

- Relatively “young” database software
  - 8 years since first release
- Known as a /dev/null database in the early days *(jepsen.io)*
  - Tremendous stability improvements since then
  - All known reliability issues has been fixed since 3.2.12
- Lots of WiredTiger bug fixes since 3.2
  - Cache eviction
  - Checkpoints
  - Lost updates and dirty writes



Version	Lost updates	Dirty Reads	Stale Reads
3.0.14	Allowed (no v1)	Allowed (no maj. read)	Allowed (no lin. read)
3.2.11	Allowed (v1 bugs)	Kinda	Allowed (no lin. read)
3.2.12	Prevented	Prevented	Allowed (no lin. read)
3.4.0-rc3	Allowed (v1 bugs)	Kinda	Kinda
3.4.0-rc4	Allowed (v1 bugs)	Kinda	Kinda
3.4.0	Prevented	Prevented	Prevented

Source: jepsen.io

~~WT-1162~~ Add latency to Jenkins wtperf tests and plots
~~WT-2026~~ Maximum pages size at eviction too large
~~WT-2221~~ Document which statistics are available via a "fast" configuration vs. an "all" configuration
~~WT-2233~~ Investigate changing when the eviction server switches to aggressive mode.
~~WT-2239~~ Make sure LSM cursors read up to date dsk_gen, it was racing with compact
~~WT-2323~~ Allocate a transaction id at the b
~~WT-2353~~ Failure to create async threads
~~WT-2380~~ Make scripts fail if code doesn't
~~WT-2486~~ Update make check so that it ru
~~WT-2555~~ make format run on Windows
~~WT-2578~~ remove write barriers from the T
~~WT-2631~~ nullptr is passed for parameters
~~WT-2638~~ ftruncate may not be supported
~~WT-2645~~ wt dump: push the complexity o
~~WT-2648~~ cache-line alignment for new pc
~~WT-2665~~ Limit allocator fragmentation in
~~WT-2678~~ The metadata should not imply
~~WT-2688~~ configure --enable-python doesn
~~WT-2693~~ Check open_cursor error paths
~~WT-2695~~ Integrate s390x accelerated crc
~~WT-2708~~ split child-update race with reco
~~WT-2711~~ Change statistics log configurati
~~WT-2719~~ add fuzz testing for WiredTiger
~~WT-2728~~ Don't re-read log file headers du
~~WT-2729~~ Focus eviction walks in largest t
~~WT-2730~~ cursor next/prev can return the
~~WT-2731~~ Raw compression can create pa
~~WT-2732~~ Coverity analysis defect 99665:
~~WT-2734~~ Improve documentation of evict
~~WT-2737~~ Scrub dirty pages rather than ev
~~WT-2738~~ Remove the ability to change th
~~WT-2739~~ pluggable file systems documer
~~WT-2743~~ Thread count statistics always r
~~WT-2744~~ partial line even with line bufferi
~~WT-2746~~ track checkpoint I/O separately
~~WT-2751~~ column-store statistics incorrect
~~WT-2752~~ Fixes to zipfian wtperf workload
~~WT-2755~~ flexelint configuration treats size
~~WT-2756~~ Upgrade the autoconf archive p
~~WT-2757~~ Column tables behave different

~~WT-2760~~ Fix a bug in backup related to directory sync. Change the filesystem API to make durab
~~WT-2762~~ wtstats tool fails if checkpoint runs
~~WT-2763~~ Unit test test_intpack failing on OSX
~~WT-2764~~ Optimize checkpoints to reduce throughput disruption
~~WT-2765~~ wt dump: indices need to be shown in the dump output
~~WT-2766~~ Don't count eviction of lookaside file pages for the purpose of checking stuck cache
~~WT-2767~~ test suite needs way to run an individual scenario
~~WT-2769~~ Update documentation to reflect correct limits of memory_page_max
~~WT-2770~~ Add statistics tracking schema operations
~~WT-2772~~ Investigate log performance testing weirdness
~~WT-2773~~ search_near in indexes does not find exact matches
~~WT-2774~~ minor cleanups/improvements
~~WT-2778~~ Python test suite: make scenario initialization consistent
~~WT-2779~~ Raw compression created unexpectedly large pages on
~~WT-2781~~ Enhance bulk cursor option with an option to return imme
~~WT-2782~~ Missing a fs_directory_list_free in ex_file_system.c
~~WT-2783~~ wtperf multi-btree.wtperf dumps core on Mac
~~WT-2785~~ Scrub dirty pages rather than evicting them: single-page
~~WT-2787~~ Include src/include/wiredtiger_ext.h is problematic
~~WT-2788~~ Java: freed memory overwrite during handle close can ca
~~WT-2791~~ Enhance OS X Evergreen unit test
~~WT-2793~~ wtperf config improvements
~~WT-2795~~ Update documentation around read-only configuration
~~WT-2796~~ Memory leak in reconciliation uncovered by stress testing
~~WT-2798~~ Crash vulnerability with nojournal after create during chec
~~WT-2800~~ Illegal file format in test/format on PPC
~~WT-2801~~ Crash vulnerability from eviction of metadata during chec
~~WT-2802~~ Transaction commit causes heap-use-after free
~~WT-2803~~ Add verbose functionality to WT Evergreen tests
~~WT-2804~~ Don't read values in a tree without a snapshot
~~WT-2805~~ Infinite recursion if error streams fail
~~WT-2806~~ wtperf allocation size off-by-one
~~WT-2807~~ Switch Jenkins performance tests to tcmalloc
~~WT-2811~~ Reconciliation asserts that transaction time has gone bac
~~WT-2812~~ Error when reconfiguring cache targets
~~WT-2813~~ small cache usage stuck even with large cache
~~WT-2814~~ Enhance wtperf to support single-op truncate mode
ice
develop, add work
g pages for evictio

~~WT-2888~~ Switch functions to return void where possible
~~WT-2892~~ hot backup can race with block truncate
~~WT-2896~~ Coverity #1362535: resource leak
~~WT-2897~~ Checkpoints can become corrupted on failure
~~WT-2901~~ Add option to disable checkpoint dirty stepdown phase
~~WT-2903~~ Reduce the impact of checkpoint scrubbing on applications
Branch: v3.2

~~WT-2818~~ The page visibility check w
~~WT-2820~~ add gcc warn_unused_res
~~WT-2822~~ panic mutex and other fun
~~WT-2823~~ support file handles withou
~~WT-2824~~ wtperf displays connection
~~WT-2826~~ clang38 false positive on u
~~WT-2827~~ checkpoint log_size config
~~WT-2828~~ Make long wtperf tests refl
~~WT-2829~~ Switch automated testing t
~~WT-2832~~ Python test uses hard-cod
~~WT-2834~~ Join cursor: discrepancy w
~~WT-2835~~ WT_CONNECTION.leak-r
~~WT-2838~~ Don't free session handles
~~WT-2839~~ lint: Ignoring return value o
~~WT-2840~~ clang analysis: garbage va
~~WT-2841~~ Jenkins Valgrind runner is
~~WT-2842~~ split wtperf's configuration
~~WT-2843~~ Fix a bug in recovery if the
~~WT-2846~~ Several bugs related to re
~~WT-2847~~ Merge fair locks into read/
~~WT-2850~~ clang 4.1 attribute warning
~~WT-2853~~ Multi threaded reader write
~~WT-2857~~ POSIX ftruncate calls shou
~~WT-2862~~ Fix lint error in test case fo
~~WT-2863~~ Support UTF-8 paths on W
~~WT-2865~~ eviction thread error failure
~~WT-2866~~ Eviction server algorithm tuning
~~WT-2867~~ Review and fix barrier usage in __lsm_tree_close
~~WT-2868~~ Add sample_interval to checkpoint-stress wtperf config
~~WT-2869~~ Performance regression on secondaries
~~WT-2870~~ Rename wtperf checkpoint schema jobs
~~WT-2871~~ __wt_verbose has the wrong GCC format attributes
~~WT-2872~~ Recent stuck cache test/stress failures.
~~WT-2873~~ Refactor CRC32 code
~~WT-2875~~ Test test_wt2853_perf can run too long under valgrind
~~WT-2876~~ Extend wtperf to support a log like table
~~WT-2878~~ Verbose changes affected performance
~~WT-2881~~ Add -Wpedantic to clang compiler warning flags
~~WT-2883~~ wiredtiger_open with verbose=handleops recursive loop
~~WT-2885~~ __wt_checkpoint_signal lint
~~WT-2886~~ Decide how in-memory configuration and eviction_dirty_targe

~~WT-2103~~ Add incremental backup testing to format
~~WT-2223~~ Add stress testing for in-memory
~~WT-2268~~ JSON load incorrect with UNICODE input
~~WT-2319~~ Add statistics around fsync calls
~~WT-2325~~ Fix an incomplete comment
~~WT-2343~~ Assert we don't remove or rename when backup cursor is open
~~WT-2349~~ Add ability to open databases read-only
~~WT-2359~~ WiredTiger with Python will hang if a calloc failure occurs during
~~WT-2360~~ Allow disjunctions and combinations of operations in join cursors
~~WT-2408~~ Windows error translation layer
~~WT-2446~~ Estimate WT cache hit ratio
~~WT-2450~~ Salvage releases pages, then explicitly evicts them.
~~WT-2453~~ Throughput drop in wtperf evict Jenkins tests
~~WT-2479~~ Dump utility discards table config (JSON)
~~WT-2491~~ The dhandle close_lock isn't valuable at the moment
~~WT-2504~~ Should READONLY always read basecfg file?
~~WT-2505~~ Review clang analyzer warnings
~~WT-2508~~ Test programs should remove test directories on the "clean" targ
~~WT-2514~~ Log path name is an empty string.
~~WT-2518~~ LSM checkpoint handle acquisition optimization
~~WT-2520~~ WT_SESSION::verify should not alter tables
~~WT-2526~~ Mixing and matching readonly and read/write handles
~~WT-2535~~ Extend test/format to test for transactions reading their writes
~~WT-2537~~ Cannot open DB written by WT2.6.1 with WT2.8.0 due to WT_N
~~WT-2539~~ Implement file streaming above pluggable filesystems
~~WT-2540~~ Separate stream and file handle methods
~~WT-2541~~ Add statistics for number of threads currently in read/write
~~WT-2542~~ Fixed-length column store reconciliation overwrites original value
~~WT-2544~~ Fix eviction statistics when clear is configured
~~WT-2546~~ Eviction server not help evict pages sometimes
~~WT-2547~~ Add 1-eviction-worker jobs to Jenkins
~~WT-2548~~ Cap the amount of data handed to raw compression.
~~WT-2549~~ joins using recno keys return no values
~~WT-2550~~ java ex_schema example fails
~~WT-2552~~ Public API for pluggable filesystems
~~WT-2553~~ Document in-memory configuration and WT_CACHE_FULL erro
~~WT-2554~~ Implement a frame
~~WT-2556~~ Typo in the Java ex
~~WT-2557~~ format test progran
~~WT-2558~~ WT_PAGE structur

~~WT-2653~~ The custom file-system example should show device configuration
~~WT-2656~~ Builds failing on GCC 4.7 builder
~~WT-2658~~ Only include PPC-specific files in PPC builds
~~WT-2659~~ csuite tests, assorted lint and cleanup.
~~WT-2660~~ Hang between eviction and connection close
~~WT-2661~~ Coverity failures: 1356050-1356053
~~WT-2662~~ For internal spell checking, strip out double quote literals, they confuse aspell
~~WT-2664~~ Change eviction so any eviction thread can find candidates
~~WT-2667~~ Enhance WiredTiger Evergreen testing
~~WT-2668~~ Create join statistics that are useful and are easy to understand
~~WT-2671~~ Dump more information about the file layout in verify debug mode
~~WT-2672~~ Handle system calls that don't set errno
~~WT-2673~~ Stop automatically increasing memory page max
~~WT-2674~~ Simplify metadata file check
~~WT-2676~~ Don't use key size in column store in-memory splits.
~~WT-2677~~ Fix JSON output so only printable ASCII is produced (seen on Solaris)
~~WT-2682~~ Add option to configure WiredTiger with strict compiler flags
~~WT-2683~~ WiredTiger no longer needs to return non-zero disk sizes
~~WT-2685~~ Hazard pointer failure from clear walk
~~WT-2686~~ Logging subsystem core dump
~~WT-2687~~ Test suite should verify the exit status of the wt utility
~~WT-2689~~ Use after free in WT_SESSION::open_cursor
~~WT-2691~~ Use wrappers for ctype functions to avoid sign extension errors
~~WT-2692~~ Fix race in file system example
~~WT-2696~~ Race condition on unclean shutdown may miss log records with large updates
~~WT-2698~~ Test/recovery hung
~~WT-2702~~ Under high thread load, WiredTiger exceeds cache size
~~WT-2704~~ Test/format hung on bengal
~~WT-2706~~ Race condition on log file switch can cause missing log records
~~WT-2707~~ dist/s_label enhancements, and error jump cleanups
~~WT-2709~~ Connection reconfigure segfault in __wt_conn_cache_pool_destroy
~~WT-2710~~ WT_FILE_HANDLE_INMEM no longer needs an off field
~~WT-2712~~ Coverity 1356928 and 1356929: ASSERT_SIDE_EFFECT
~~WT-2713~~ Document WT_PANIC so pluggable filesystems can panic.
~~WT-2714~~ Lint
~~WT-2715~~ random-abort test may write partial record at the end
~~WT-2720~~ Pull request tester not running Python suite
~~WT-2722~~ s_label or s_label_loop false positive
~~WT-2724~~ Eviction workers created on open exit immediately
~~WT-2763~~ Unit test test_intpack failing on OSX

~~WT-2864~~ Reconfiguring the checkpoint server can lead to hangs
~~WT-2874~~ Change test_compact01 to avoid eviction
~~WT-2918~~ The dist scripts create C files s_whitespace complains about
~~WT-2919~~ Don't mask error returns from style checking scripts
~~WT-2921~~ Reduce the WT_SESSION hazard_size when possible
~~WT-2923~~ heap-use-after-free on address in compaction
~~WT-2924~~ Ensure we are doing eviction when threads are waiting for it
~~WT-2925~~ WT_THREAD_PANIC_FAIL is a WT_THREAD structure flag
~~WT-2926~~ WT_CONNECTION.reconfigure can attempt unlock of not-locked lock
~~WT-2928~~ Eviction failing to switch queues can lead to starvation

MongoDB

- Everything comes as standard:
    - Built-in replication
    - Built-in sharding
    - Live cluster version upgrades *(ish)*
        - Shutdown slave, upgrade slave, startup slave, repeat

Server Hardware

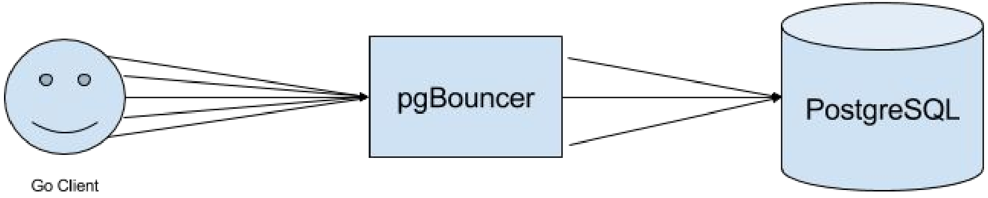
- 2x Intel(R) Xeon(R) CPU E5-2630 v4
  - 20 cores / 40 threads
- 32GB Memory
- FreeBSD 11
- ZFS file system
- 2 x 1.6TB (Intel SSD DC S3610, MLC)

Why do we use ZFS?

- Highly tunable filesystem
  - Layered caching (ARC, L2ARC, ZIL)
  - Advanced cache algorithm
    - Most Recently Used (MRU)
    - Most Frequently Used (MFU)
- Free snapshots
- Block level checksums
- Fast compression
- Nexenta, Delphix, Datto, Joyent, Tegile, Oracle (*obviously*) and many more!

The Setup

- 1-3 Client machines (depending on the test)
- 1 Server, two jails - one for Postgres & one for Mongo
- PostgreSQL 9.6.5 with pgBouncer 1.7.2
- MongoDB 3.4.9 with WiredTiger



Performance Tuning

- We had to tune PostgreSQL heavily
  - System V IPC (shmmax, shmall, semmns and etc)
  - pgBouncer (single threaded, we need multiple instances to handle the load)

- MongoDB tuning was easy!
  - WiredTiger cache size
  - Compression settings
  - Default settings are usually good enough

- ZFS tuning
  - atime
  - recordsize
  - checksum
  - compression

Sample JSON Document

```
{
	"_id" : NumberLong(2),
	"name" : "lPAyAYpUvUDGiCd",
	"addresses" : [
		{
			"number" : 59,
			"line1" : "EPJKLhmEPrrdYqaFxxEVMF",
			"line2" : "Rvlgkmb"
		},
		{
			"number" : 59,
			"line1" : "DdCBXEW",
			"line2" : "FEV"
		}
	],
	"phone_number" : "xPOYCOfSpieIxbGxpYEpi",
	"dob" : ISODate("2017-09-05T00:03:28.956Z"),
	"age" : 442006075,
	"balance" : 0.807247519493103,
	"enabled" : false,
	"counter" : 442006075,
	"padding" : BinData(0,"")
}
```

Sub-documents

Varying field types

Randomised binary blob

About Me

- Engineer on the High Performance Platforms team
  - Our team builds a high volume CA platform & distributed systems
  - Based in Old Street, London
  - Greenfields project, all new stuff!

- Day job has me breaking all the things
  - Simulating failures, network partitions, etc
  - Assessing performance and durability

- Maintain performance fork of Go MongoDB driver
  - github.com/globalsign/mgo

MPJBT Benchmark Tool

- MongoDB PostgreSQL JSONB Benchmarking Tool
  - *Seriously*, we’re open to better names…….
- Written in Golang
- Open source!
- Models typical workloads (but maybe not yours!)
  - Inserts, selects, select-updates, range queries, etc.
- Lockless outside of the database drivers
  - Low contention improves ability to push servers

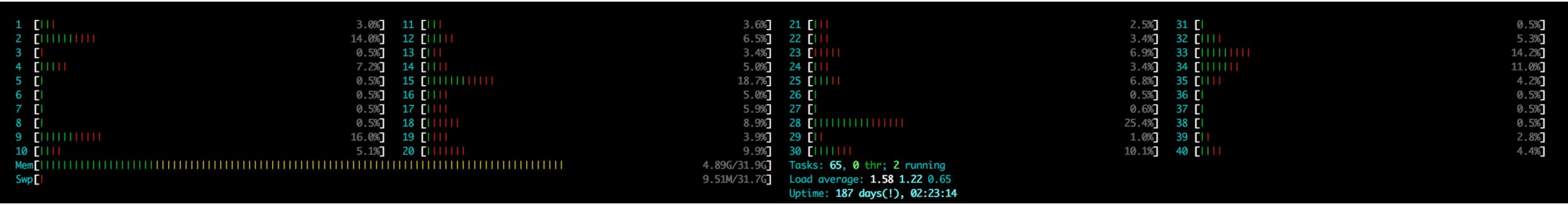
Why Go?

- Designed from the start for high concurrency
    - Thousands of concurrent workers is totally fine

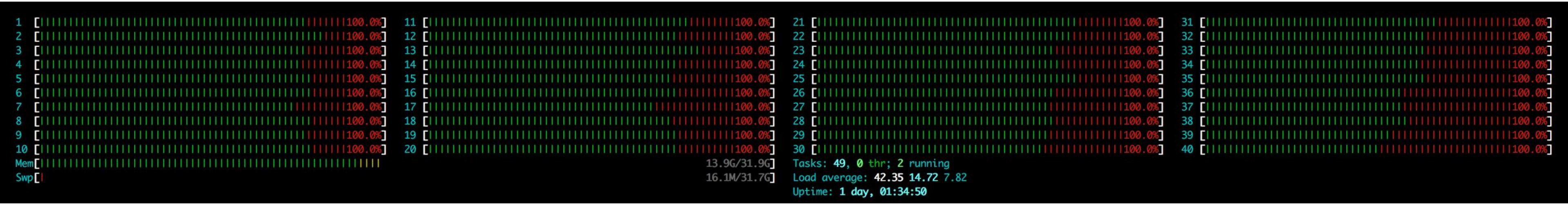
- Co-operative scheduler can maximise I/O throughput
    - When blocked, Go switches to another worker
    - Blocked worker is woken up when it’s unblocked
    - Much cheaper context switching - occurs in userland

- Familiarity - I use it every day!

Does it deliver?

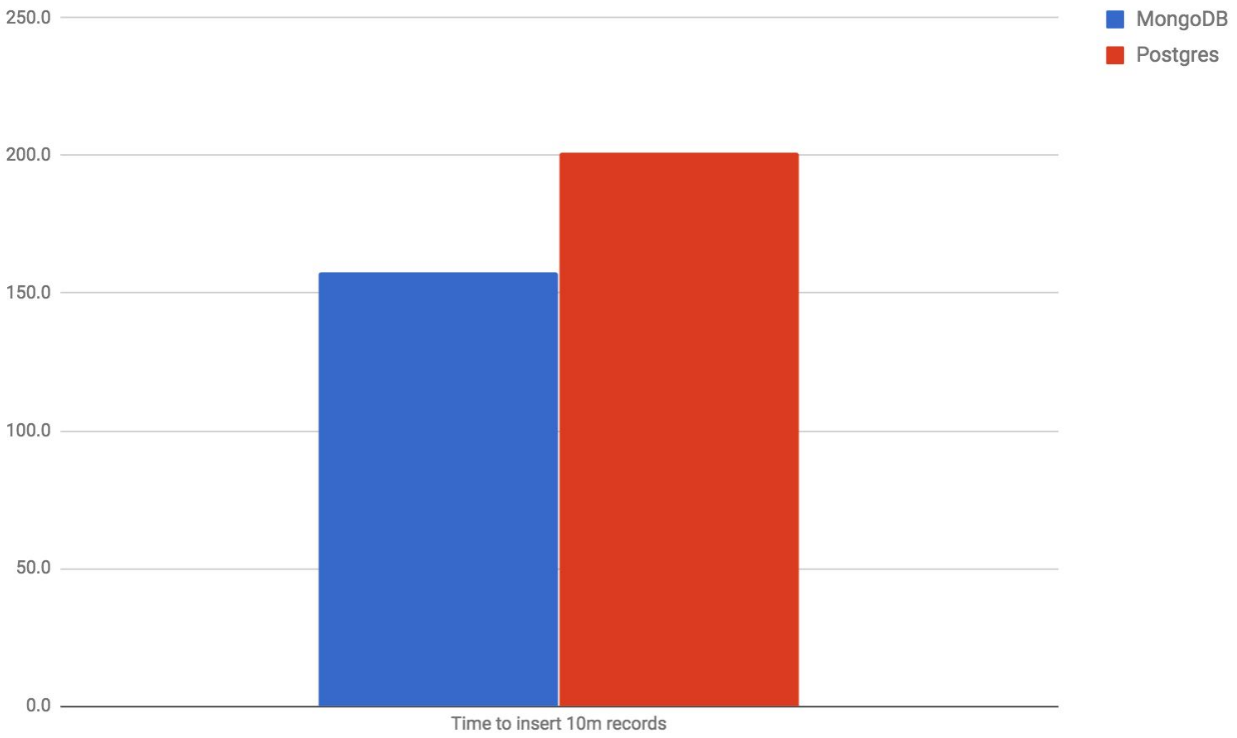


```
 1 [|||                   3.0%] 11 [|||                   3.6%] 21 [|||                   2.5%] 31 [|                     0.5%]
 2 [||||||||||           14.0%] 12 [|||||                 6.5%] 22 [|||                   3.4%] 32 [||||                  5.3%]
 3 [|                     0.5%] 13 [|||                   3.4%] 23 [|||||                 6.9%] 33 [|||||||||            14.2%]
 4 [|||||                 7.2%] 14 [||||                  5.0%] 24 [|||                   3.4%] 34 [|||||||              11.0%]
 5 [|                     0.5%] 15 [||||||||||||         18.7%] 25 [|||||                 6.8%] 35 [||||                  4.2%]
 6 [|                     0.5%] 16 [||||                  5.0%] 26 [|                     0.5%] 36 [|                     0.5%]
 7 [|                     0.5%] 17 [||||                  5.9%] 27 [|                     0.6%] 37 [|                     0.5%]
 8 [|                     0.5%] 18 [||||||                8.9%] 28 [||||||||||||||||     25.4%] 38 [|                     0.5%]
 9 [|||||||||||          16.0%] 19 [||||                  3.9%] 29 [||                    1.0%] 39 [||                    2.8%]
10 [||||                  5.1%] 20 [|||||||               9.9%] 30 [|||||||              10.1%] 40 [||||                  4.4%]
Mem[||||||||||||||||||||||||||||||||||||||||||||||   4.89G/31.9G] Tasks: 65, 0 thr; 2 running
Swp[|                                                9.51M/31.7G] Load average: 1.58 1.22 0.65
                                                                  Uptime: 187 days(!), 02:23:14
```



```
 1 [||||||||||||||||||||||100.0%] 11 [||||||||||||||||||||||100.0%] 21 [||||||||||||||||||||||100.0%] 31 [||||||||||||||||||||||100.0%]
 2 [||||||||||||||||||||||100.0%] 12 [||||||||||||||||||||||100.0%] 22 [||||||||||||||||||||||100.0%] 32 [||||||||||||||||||||||100.0%]
 3 [||||||||||||||||||||||100.0%] 13 [||||||||||||||||||||||100.0%] 23 [||||||||||||||||||||||100.0%] 33 [||||||||||||||||||||||100.0%]
 4 [||||||||||||||||||||||100.0%] 14 [||||||||||||||||||||||100.0%] 24 [||||||||||||||||||||||100.0%] 34 [||||||||||||||||||||||100.0%]
 5 [||||||||||||||||||||||100.0%] 15 [||||||||||||||||||||||100.0%] 25 [||||||||||||||||||||||100.0%] 35 [||||||||||||||||||||||100.0%]
 6 [||||||||||||||||||||||100.0%] 16 [||||||||||||||||||||||100.0%] 26 [||||||||||||||||||||||100.0%] 36 [||||||||||||||||||||||100.0%]
 7 [||||||||||||||||||||||100.0%] 17 [||||||||||||||||||||||100.0%] 27 [||||||||||||||||||||||100.0%] 37 [||||||||||||||||||||||100.0%]
 8 [||||||||||||||||||||||100.0%] 18 [||||||||||||||||||||||100.0%] 28 [||||||||||||||||||||||100.0%] 38 [||||||||||||||||||||||100.0%]
 9 [||||||||||||||||||||||100.0%] 19 [||||||||||||||||||||||100.0%] 29 [||||||||||||||||||||||100.0%] 39 [||||||||||||||||||||||100.0%]
10 [||||||||||||||||||||||100.0%] 20 [||||||||||||||||||||||100.0%] 30 [||||||||||||||||||||||100.0%] 40 [||||||||||||||||||||||100.0%]
Mem[||||||||||||||||||||||||||||||                   13.9G/31.9G] Tasks: 49, 0 thr; 2 running
Swp[|                                                16.1M/31.7G] Load average: 42.35 14.72 7.82
                                                                  Uptime: 1 day, 01:34:50
```

Insert 10,000,000 records



Average isn’t very helpful

- I have an average of 52.2ms

Average isn’t very helpful

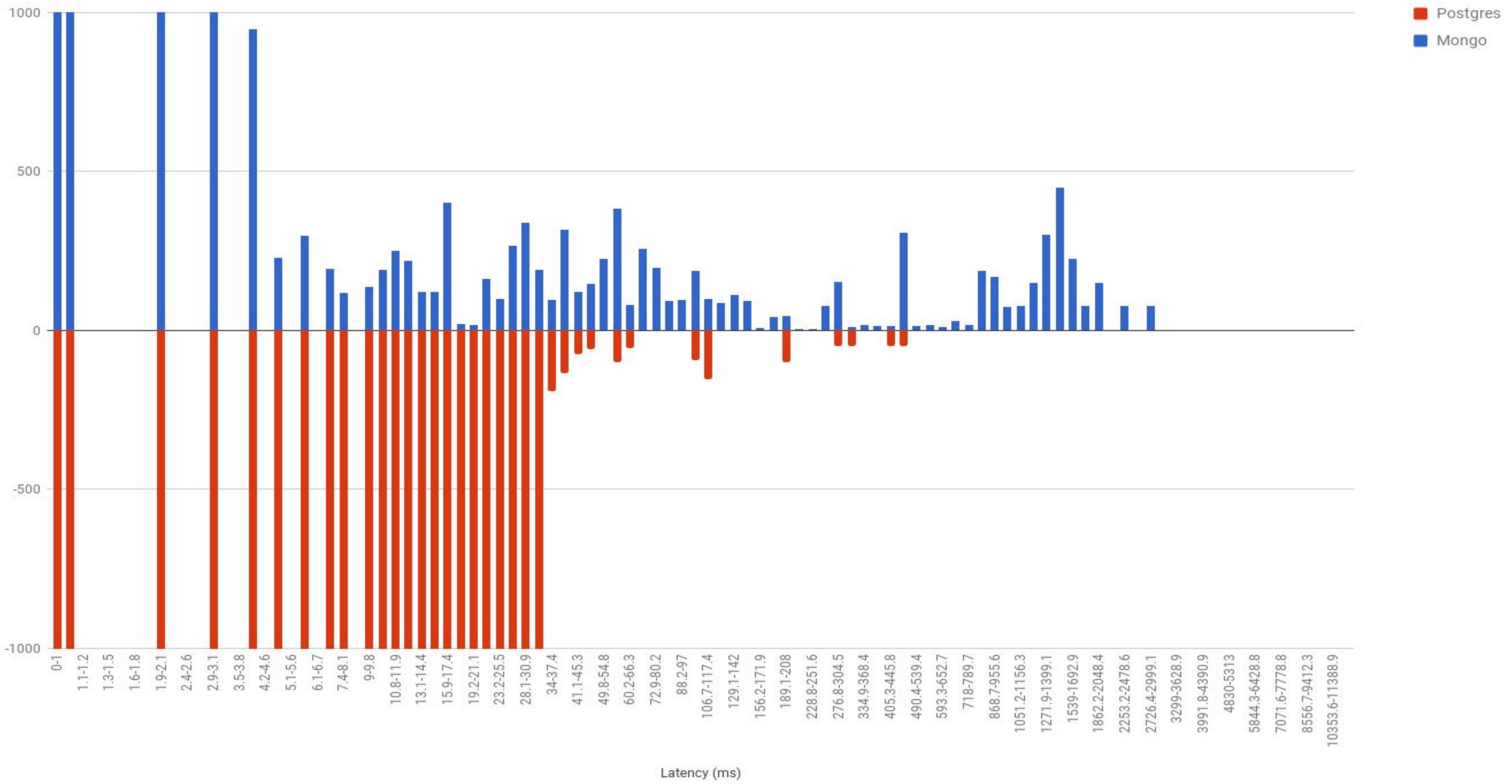
- I have an average of 52.2ms

```
120.080231
 36.237584
 25.904811
 44.053916
 66.617778
 59.713100
 74.620329
  1.689589
 90.641940
 27.202953
```

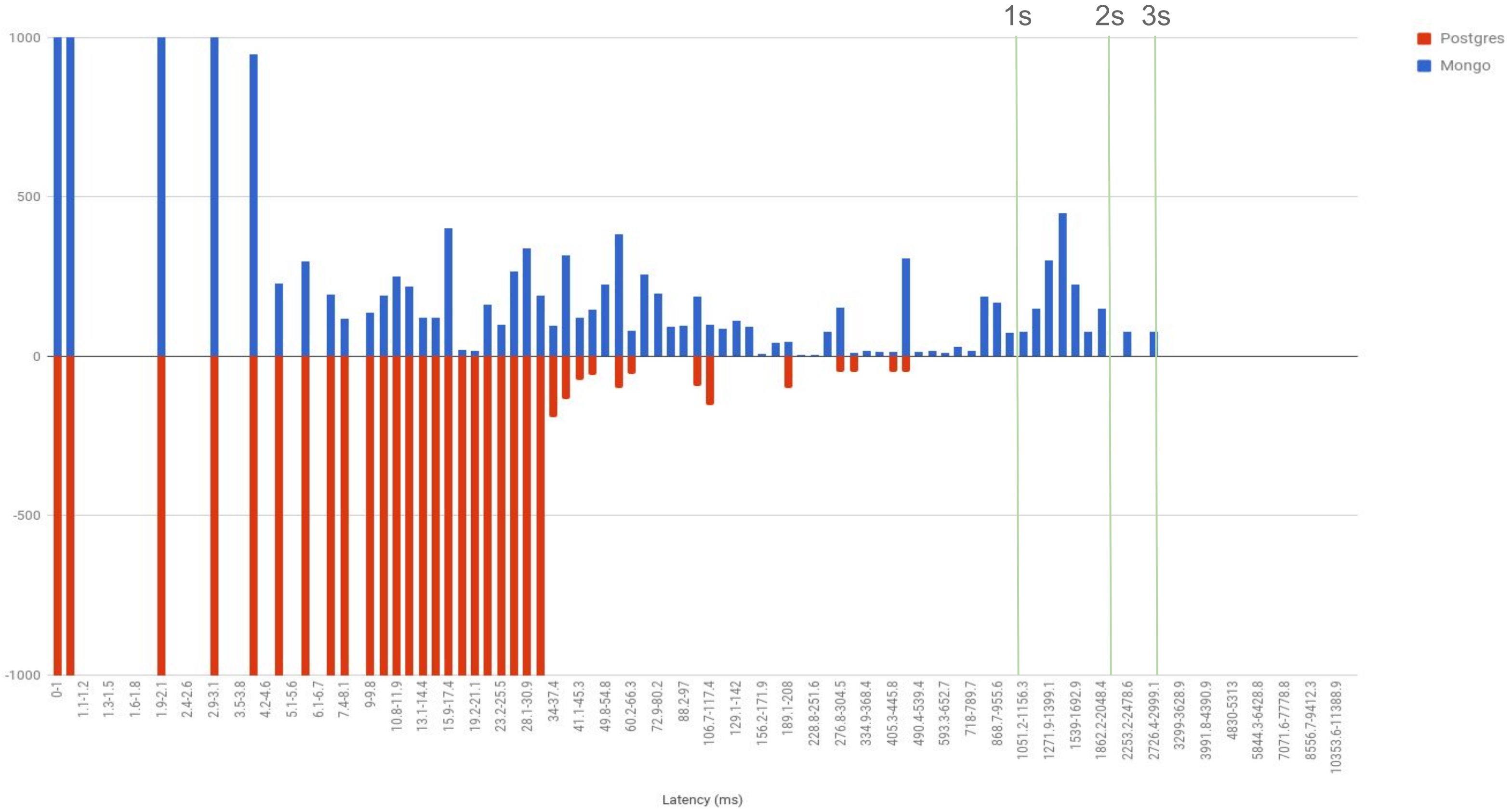
OR

```
51.162331
52.202392
52.511745
50.439697
52.975609
52.567941
53.067609
52.122890
51.159180
52.390616
```

Inserts - Latency Histogram



Inserts - Latency Histogram

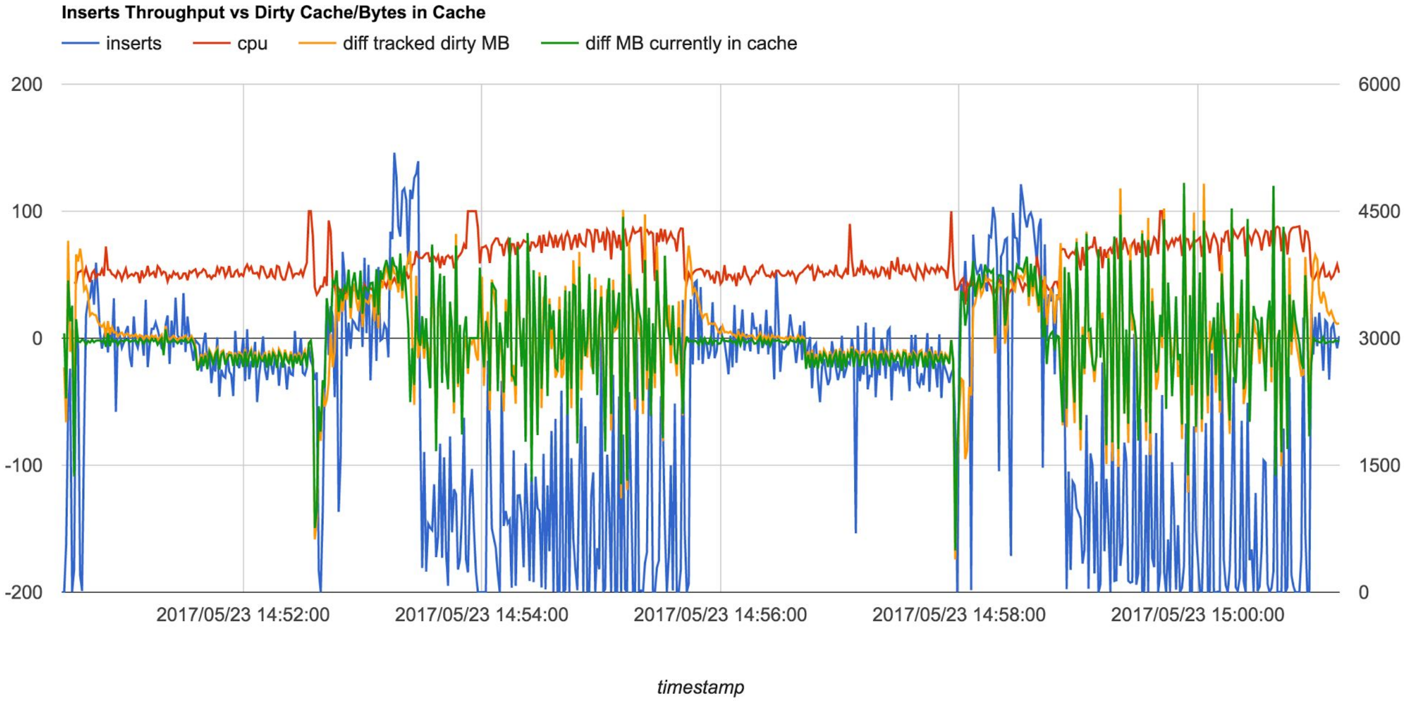


Inserts - Throughput

```
insert 30877op/s avg.0ms
insert 27509op/s avg.0ms
insert 29997op/s avg.0ms
insert 31143op/s avg.0ms
insert 22576op/s avg.0ms
insert 0op/s avg.0ms
insert 0op/s avg.0ms
insert 1op/s avg.2561ms
insert 0op/s avg.0ms
insert 20703op/s avg.6ms
insert 31154op/s avg.0ms
insert 31298op/s avg.0ms
insert 30359op/s avg.0ms
```

```
insert 26081op/s avg.0ms
insert 25938op/s avg.0ms
insert 26649op/s avg.0ms
insert 26009op/s avg.0ms
insert 26029op/s avg.0ms
insert 25522op/s avg.0ms
insert 25960op/s avg.0ms
insert 26000op/s avg.0ms
insert 25576op/s avg.0ms
insert 26159op/s avg.0ms
insert 25628op/s avg.0ms
insert 26071op/s avg.0ms
insert 25856op/s avg.0ms
```

MongoDB cache eviction bug?



MongoDB cache eviction bug - not a bug?

- Reported to MongoDB *(twice!)*
  - https://jira.mongodb.org/browse/SERVER-29311
  - Offered to run any tests and analyse data
- Ran 36 different test combinations
  - ZFS compression: lz4, zlib, off
  - MongoDB compression: snappy, zlib, off
  - Filesystem block sizes
  - Disk configurations
  - Tried running on Linux/XFS
- Always saw the same pauses
  - Described as an I/O bottleneck

Profile with Dtrace!

- Dynamic tracer built into FreeBSD (and others)
  - Originally created by Sun for Solaris
  - Ported to FreeBSD
  - Low profiling overhead

- Traces in both kernel and userspace
  - Hook syscalls, libc, application functions, etc
  - Access function arguments, kernel structures, etc

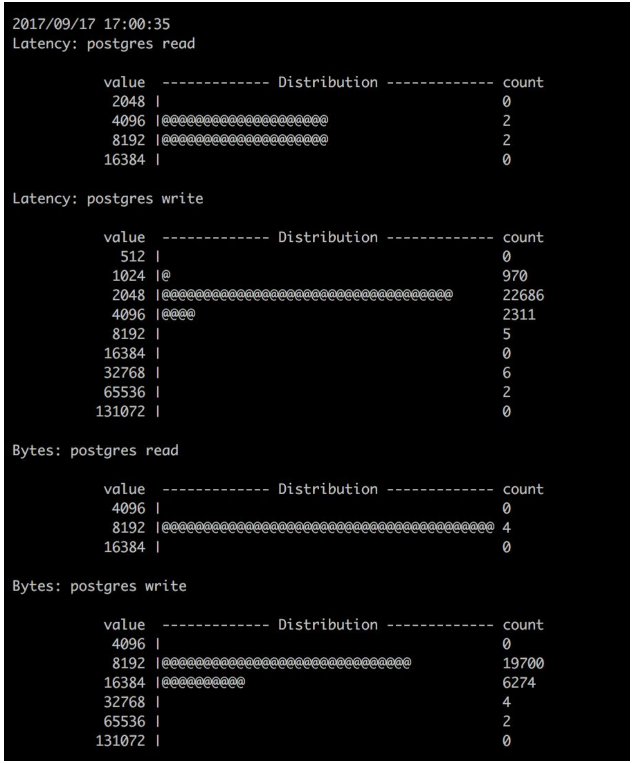
- Hooks expressed in D like DSL
  - Conditionally trigger traces
  - Really simple to use

Trace the Virtual File System

- Measures application file system operations
  - Kernel level
  - File system agnostic (XFS, ZFS, anything)

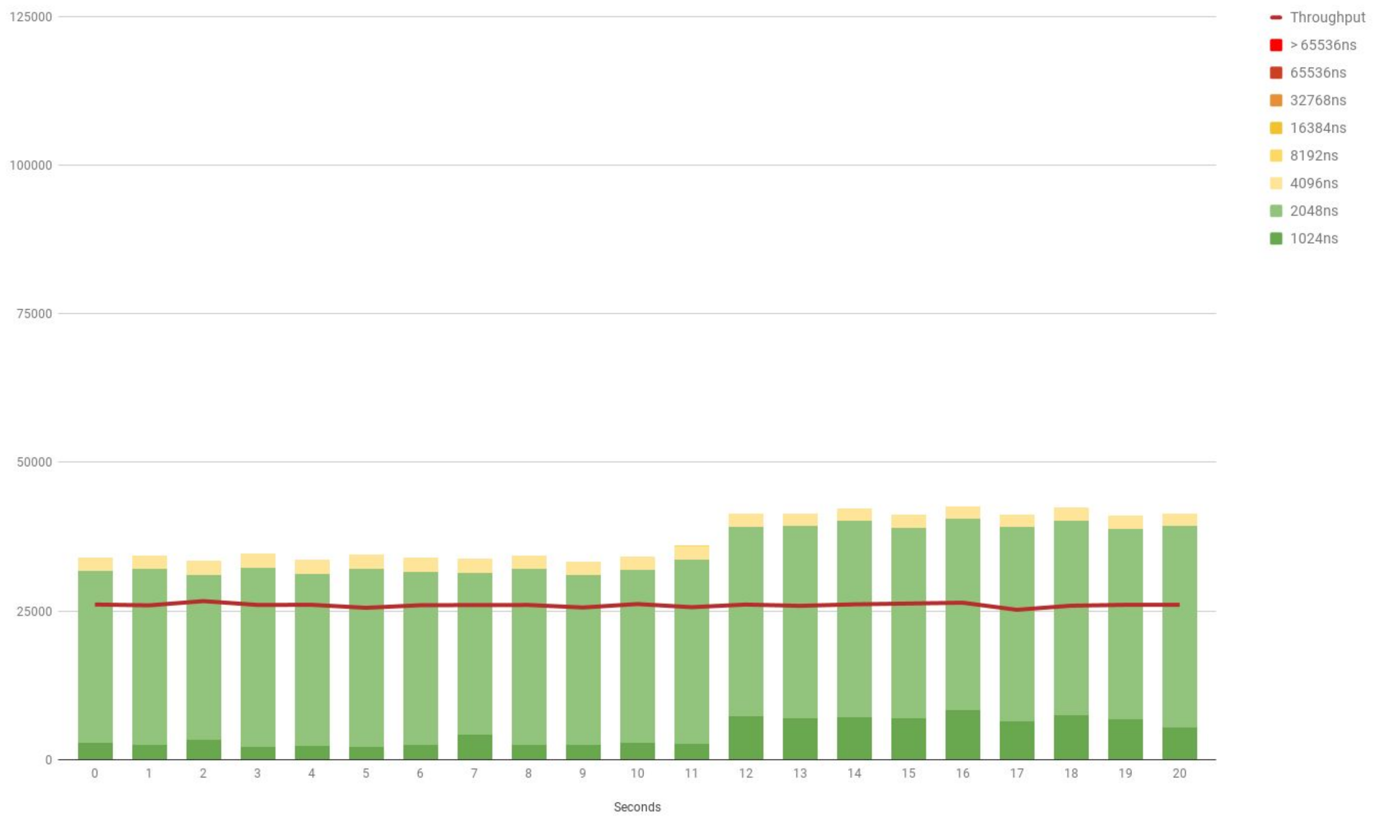
- Records data size & latency:
  - Reads - `vfs::vop_read`
  - Writes - `vfs::vop_write`

- Configured to output ASCII histograms
  - Per second aggregations
  - Broken down by type
  - Timestamped for correlating with MPJBT logs

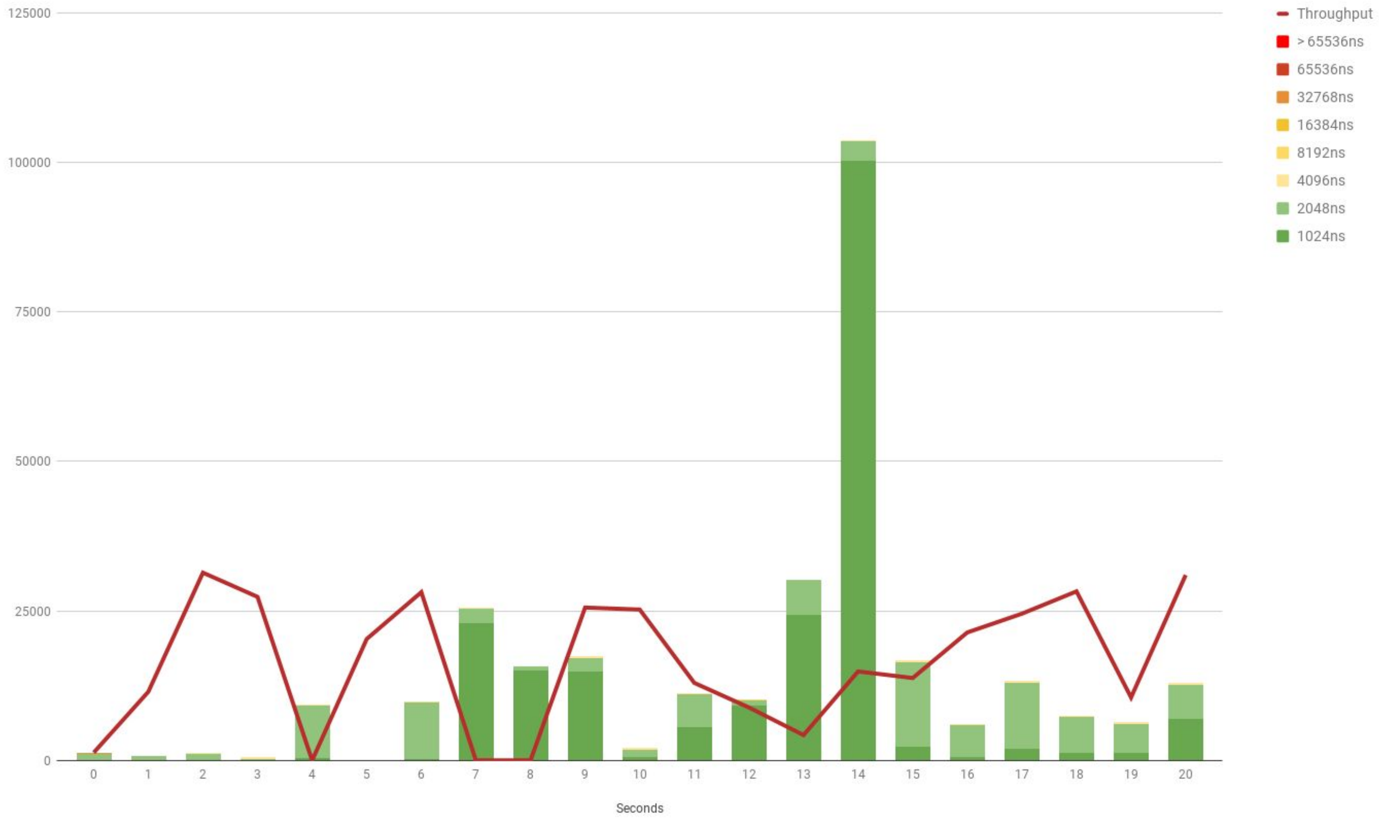


```
2017/09/17 17:00:35
Latency: postgres read

           value  ------------- Distribution ------------- count
            2048 |                                         0
            4096 |@@@@@@@@@@@@@@@@@@@@                     2
            8192 |@@@@@@@@@@@@@@@@@@@@                     2
           16384 |                                         0

Latency: postgres write

           value  ------------- Distribution ------------- count
             512 |                                         0
            1024 |@                                        970
            2048 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@      22686
            4096 |@@@@                                     2311
            8192 |                                         5
           16384 |                                         0
           32768 |                                         6
           65536 |                                         2
          131072 |                                         0

Bytes: postgres read

           value  ------------- Distribution ------------- count
            4096 |                                         0
            8192 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 4
           16384 |                                         0

Bytes: postgres write

           value  ------------- Distribution ------------- count
            4096 |                                         0
            8192 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@           19700
           16384 |@@@@@@@@@@                               6274
           32768 |                                         4
           65536 |                                         2
          131072 |                                         0
```

VFS Writes vs. Throughput - PostgreSQL



VFS Writes vs. Throughput - MongoDB



Insert / Update / Select comparison

- Preloaded 10,000,000 records in the table
  - No padding - records are ~320 bytes

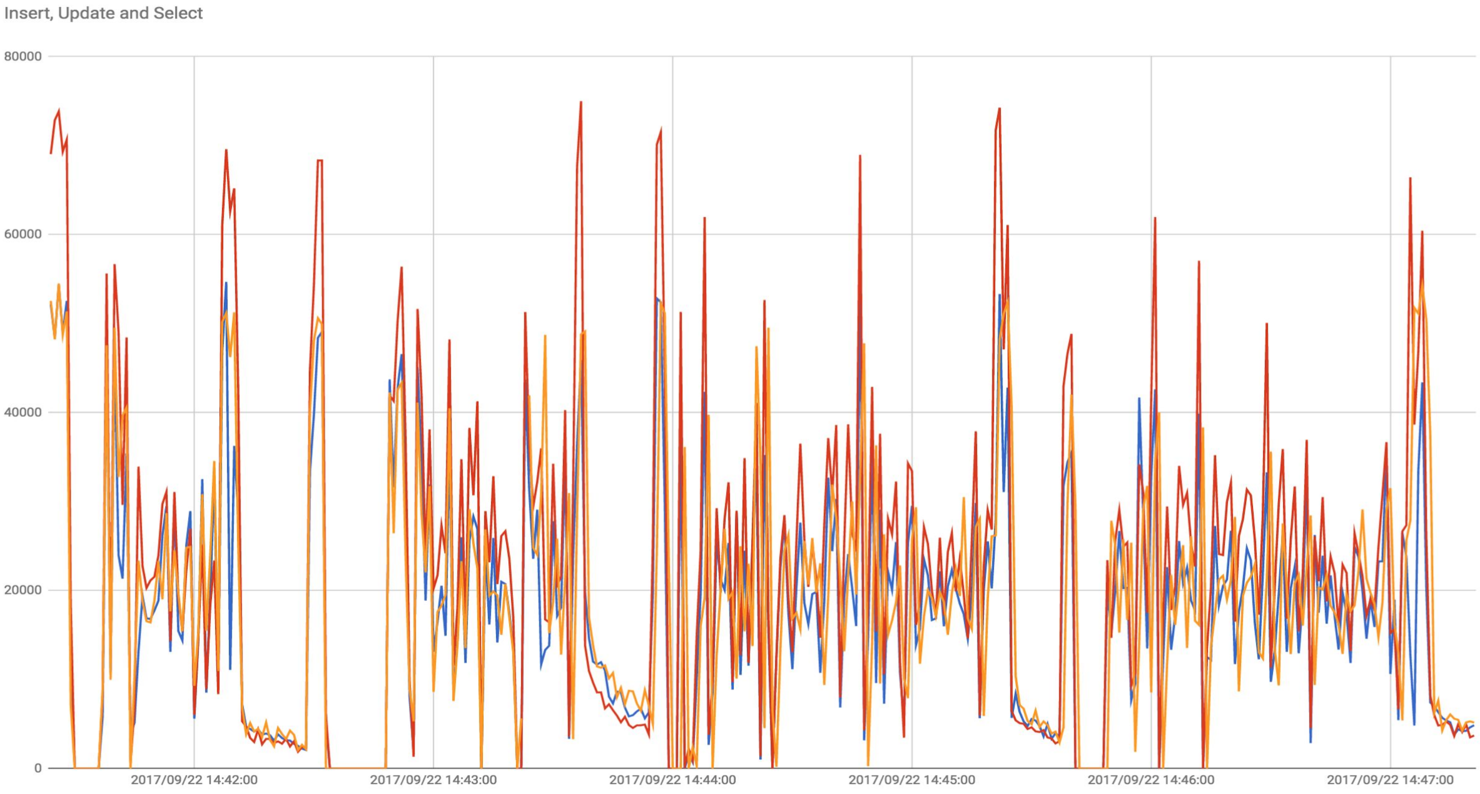
- 3 clients running different workloads
  - 50 workers inserting
  - 50 workers updating
  - 50 workers performing a range over partial index

- Both databases become CPU bound
  - Database server is under maximum load
  - Typically avoided in a production environment
  - Always good to know your maximum numbers

MongoDB

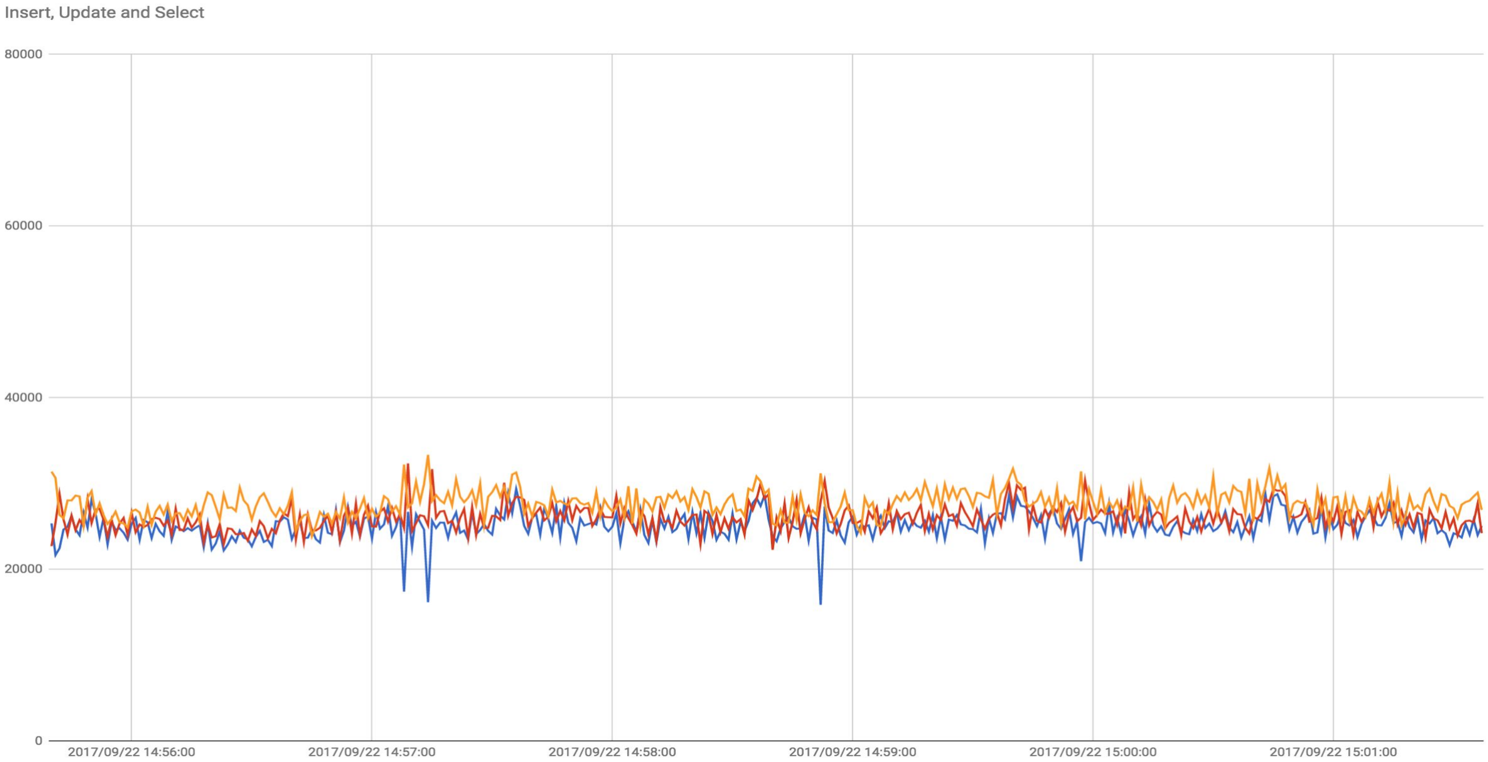
Insert	**Update**	**Select**
99th%	99th%	99th%
13ms	**11ms**	**12ms**
Average	Average	Average
18,070 op/s	**22,304 op/s**	**18,960 op/s**

MongoDB


Insert, Update and Select
80000
60000
40000
20000
0
2017/09/22 14:42:00
2017/09/22 14:43:00
2017/09/22 14:44:00
2017/09/22 14:45:00
2017/09/22 14:46:00
2017/09/22 14:47:00

I
LOVE
[redacted]INGO
MO

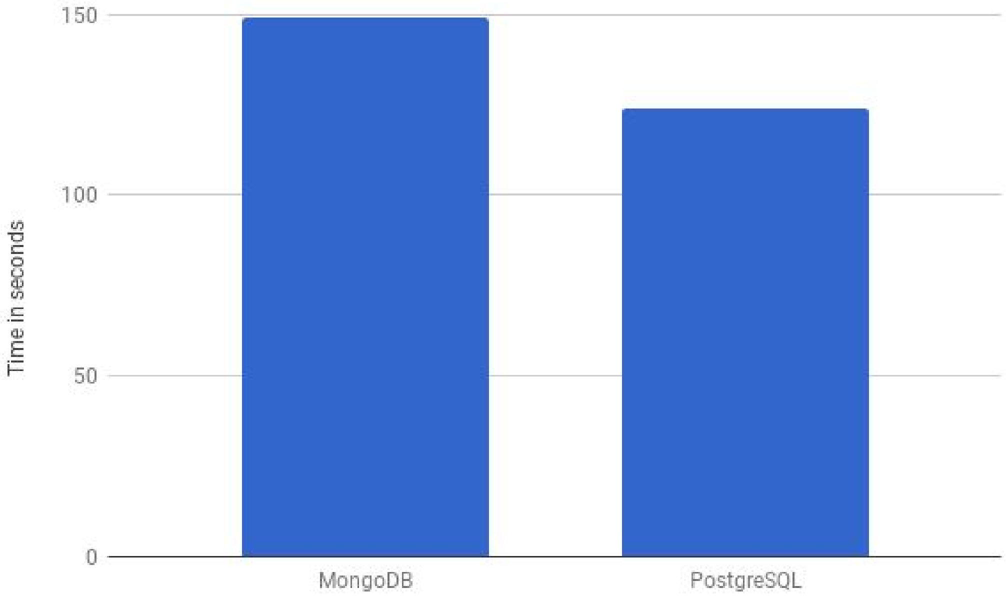
PostgreSQL


Insert, Update and Select
80000
60000
40000
20000
0
2017/09/22 14:56:00
2017/09/22 14:57:00
2017/09/22 14:58:00
2017/09/22 14:59:00
2017/09/22 15:00:00
2017/09/22 15:01:00

PostgreSQL

Insert	**Update**	**Select**
99th%	99th%	99th%
4ms	**4ms**	**3ms**
Average	Average	Average
25,244 op/s	**26,085 op/s**	**27,778 op/s**

Workload - 1MB Inserts



Lower is better

99th Percentile Latency

- MongoDB => **4.5s**
- PostgreSQL => 1.5s

Insert Performance

CPU 35%

```
insert 65543op/s avg.0ms
insert 65113op/s avg.0ms
insert 69881op/s avg.0ms
insert 55728op/s avg.0ms
insert 57502op/s avg.0ms
insert 64428op/s avg.0ms
insert 64872op/s avg.6ms
insert 68804op/s avg.0ms
insert 63204op/s avg.0ms
insert 63279op/s avg.0ms
```

CPU 40%

```
insert 42011op/s avg.0ms
insert 53330op/s avg.0ms
insert 57815op/s avg.0ms
insert 54331op/s avg.0ms
insert 39616op/s avg.0ms
insert 51919op/s avg.0ms
insert 53366op/s avg.0ms
insert 56678op/s avg.0ms
insert 40283op/s avg.0ms
insert 47300op/s avg.0ms
```

Update Performance

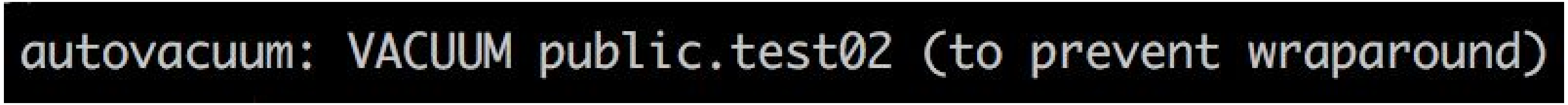
CPU 85%

```
update 2416 op/s avg.0ms
update 0    op/s avg.0ms
update 0    op/s avg.0ms
update 2856 op/s avg.33ms
update 21425op/s avg.0ms
update 0    op/s avg.0ms
update 0    op/s avg.0ms
update 12798op/s avg.5ms
update 11094op/s avg.0ms
update 21302op/s avg.0ms
```

CPU 65%

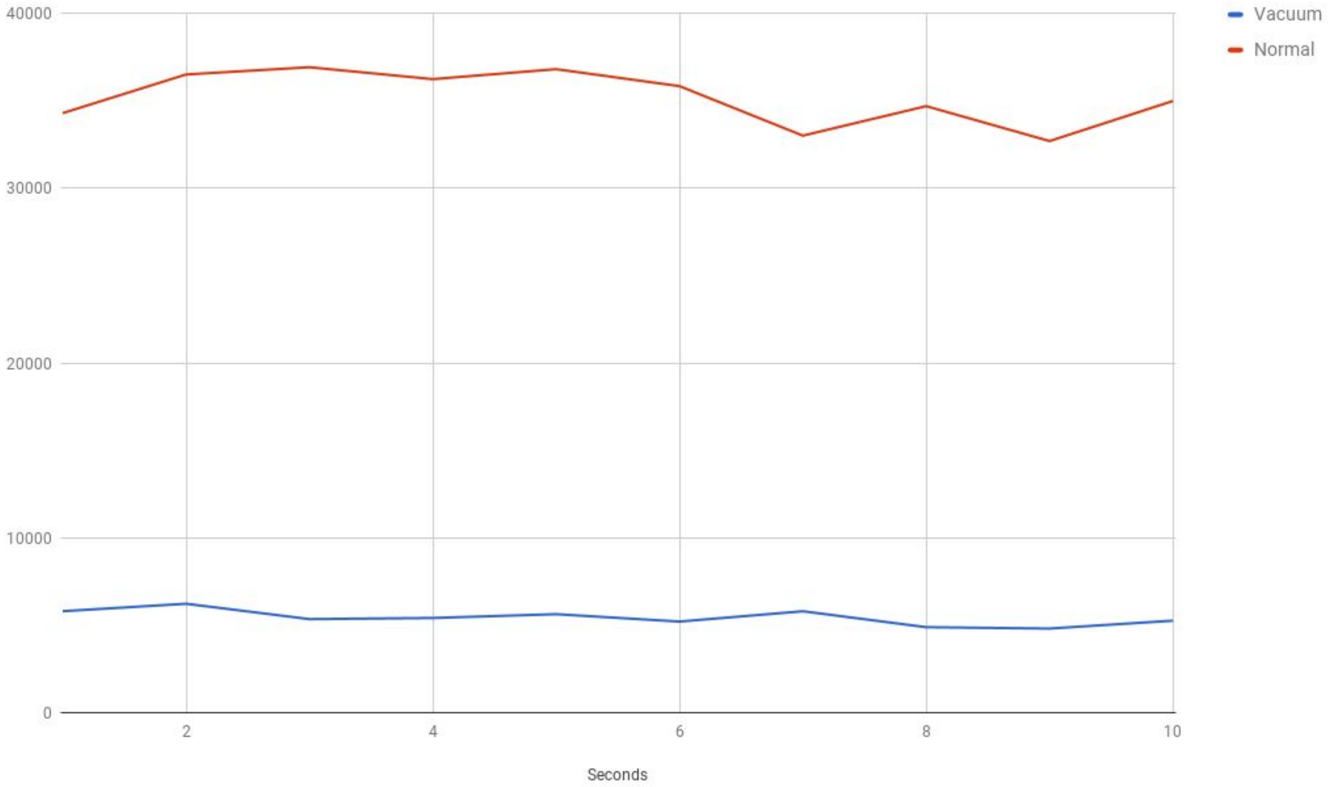
```
update 31252op/s avg.0ms
update 32706op/s avg.0ms
update 33801op/s avg.0ms
update 28276op/s avg.0ms
update 34749op/s avg.0ms
update 29972op/s avg.0ms
update 28565op/s avg.0ms
update 32286op/s avg.0ms
update 30905op/s avg.0ms
update 32052op/s avg.0ms
```

So...why are we even using MongoDB?
PostgreSQL is awesome right?



```
autovacuum: VACUUM public.test02 (to prevent wraparound)
```

Vacuum Performance - Insert Workload



Horizontally Scalable

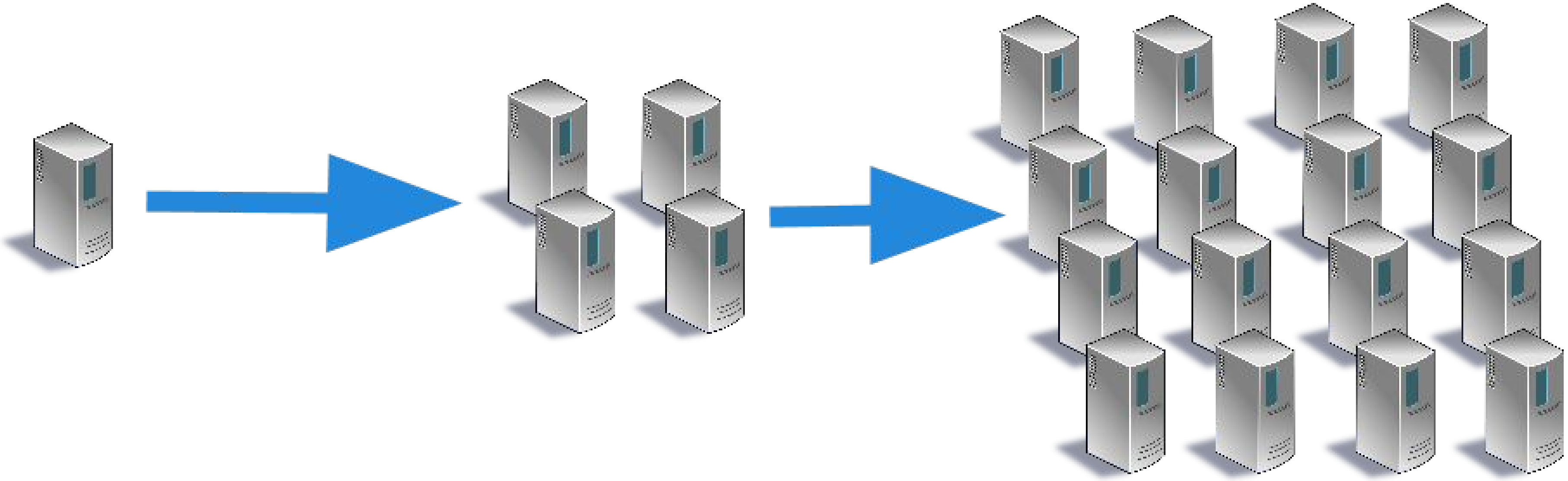
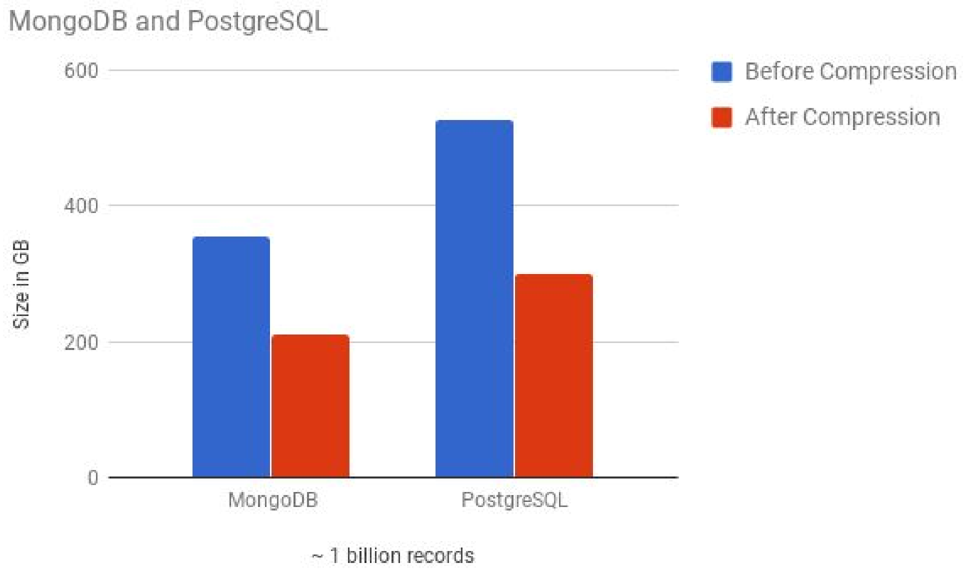
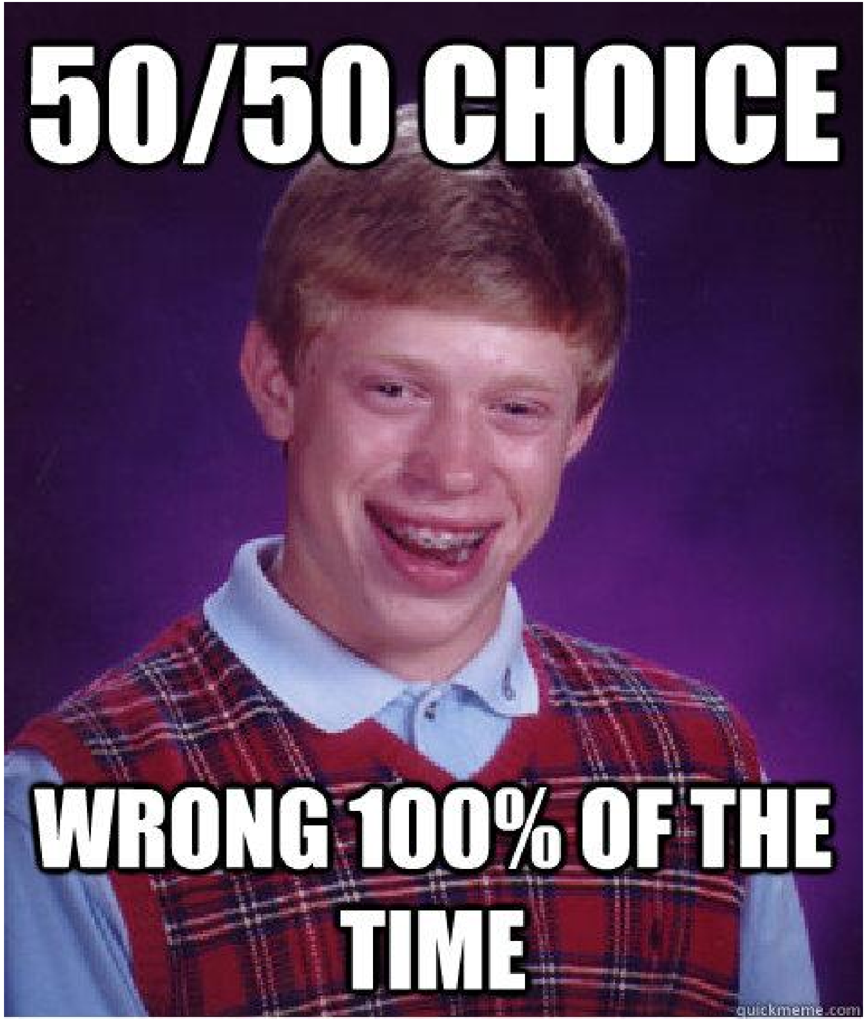


Table Size Comparison



Lower is better

Summary


50/50 CHOICE
WRONG 100% OF THE TIME
quickmeme.com

Summary

- There is no such thing as the best database in the world!
- Choosing the right database for your application is never easy
    - How well does it scale?
    - How easy is it to perform upgrades?
    - How does it behave under stress?
- What is your application requirements?
    - Do you really *need* ACID?
- Do your own research!

Summary - PostgreSQL

- PostgreSQL has poor performance out of the box
  - Requires a decent amount of tuning to get good performance out of it
- Does not scale well with large number of connections
  - pgBouncer is a must
- Combines ACID compliance with schemaless JSON
- Queries not really intuitive

Summary - MongoDB

- MongoDB has decent performance out of the box.
- Unstable throughput and latency
- Scale well with large number of connections
- Strong horizontal scalability
- Throughput bug is annoying
- MongoDB rolling upgrades are ridiculously easy
- Developer friendly - easy to use!

TODO

- Released MPJBT on Github
  - Open source for all
  - github.com/domodwyer/mpjbt
- Run similar tests against CitusDB
  - You guys have inspired us to keep looking!
- Run performance test for MongoRocks (LSM)

Questions?

Thank You!

Like what you see?
We are hiring!
Come and speak to us!

GlobalSign®
GMO INTERNET GROUP

References

- https://people.freebsd.org/~seanc/postgresql/scale15x-2017-postgresql_zfs_best_practices.pdf
- https://jepsen.io/analyses/mongodb-3-4-0-rc3
- https://dba.stackexchange.com/questions/167525/inconsistent-statistics-on-jsonb-column-with-btree-index
- https://github.com/domodwyer/mpjbt
- https://jira.mongodb.org/browse/WT-3633

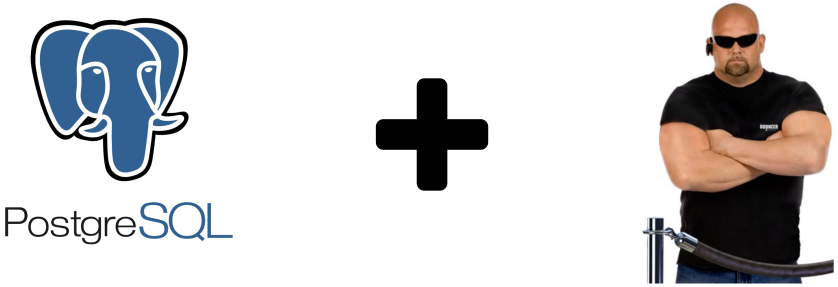
Previous Benchmark Results

- http://tiborsimko.org/postgresql-mongodb-json-select-speed.html
- http://erthalion.info/2015/12/29/json-benchmarks/
- https://www.enterprisedb.com/postgres-plus-edb-blog/marc-linster/postgres-outperforms-mongodb-and-ushers-new-developer-reality
- https://pgconf.ru/media/2017/04/03/20170317H2_O.Bartunov_json-2017.pdf
- https://www.slideshare.net/toshiharada/ycsb-jsonb

Appendix

pgbouncer.ini

- PostgreSQL does not support connection pooling
- PgBouncer is an extremely lightweight connection pooler
- Setting up and tearing down a new connection is expensive
- Each PostgreSQL connection forks a new process
- Configuration
  - pool_mode = transaction
  - max_client_conn = 300



postgresql.conf

- shared_buffer = 16GB
- max_connections = 400
- fsync = on
- synchronous_commit = on
- full_page_writes = off
- wal_compression = off
- wal_buffers = 16MB
- min_wal_size = 2GB
- max_wal_size = 4GB
- checkpoint_completion_target = 0.9
- work_mem = 33554KB
- maintenance_work_mem = 2GB
- wal_level=replica

mongod.conf

- wiredTiger.engineConfig.cacheSizeGB: 19
- wiredTiger.engineConfig.journalCompressor: snappy
- wiredTiger.collectionConfig.blockCompressor: snappy
- wiredTiger.indexConfig.prefixCompression: true
- net.maxIncomingConnections: 65536
- wiredTigerConcurrentReadTransactions: 256
- wiredTigerConcurrentWriteTransactions: 256

ZFS Tuning

- No separate L2ARC
- No separate ZIL
- 1 dataset for O/S
- 1 dataset for data directory
  - checksum=on
  - atime=off
  - recordsize=8K
  - compression=lz4 (PostgreSQL) or off (MongoDB)

/boot/loader.conf

- kern.maxusers=1024
- kern.ipc.semmns=2048
- kern.ipc.semmni=1024
- kern.ipc.semmnu=1024
- kern.ipc.shmall=34359738368
- kern.ipc.shmmax=34359738368
- kern.ipc.maxsockets=256000
- kern.ipc.maxsockbuf=2621440
- kern.ipc.shmseg=1024

/etc/sysctl.conf

- net.inet.tcp.keepidle=3000000
- net.inet.tcp.keepintvl=60000
- net.inet.tcp.keepinit=60000
- security.jail.sysvipc_allowed=1
- kern.ipc.shmmax=34359738368
- kern.ipc.shmall=16777216
- kern.ipc.shm_use_phys=1
- kern.maxfiles=2621440
- kern.maxfilesperproc=2621440
- kern.threads.max_threads_per_proc=65535
- kern.ipc.somaxconn=65535
- kern.eventtimer.timer=HPET
- kern.timecounter.hardware=HPET
- vfs.zfs.arc_max: 8589934592 for PostgreSQL or 1073741824 for MongoDB