

# POSTGRESQL AT LOW LEVEL

STAY CURIOUS!

DMITRY DOLGOV

07-03-2019





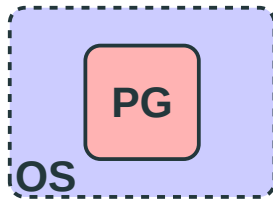
patroni & postgres-operator

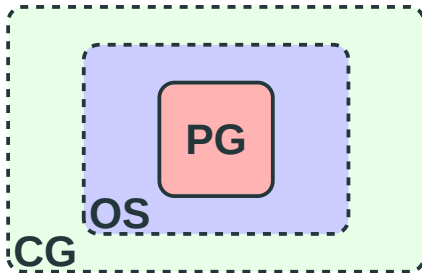
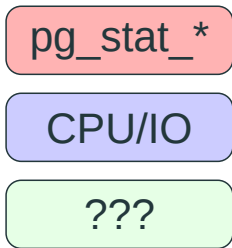
pg\_stat\_\*

PG

pg\_stat\_\*

CPU/IO



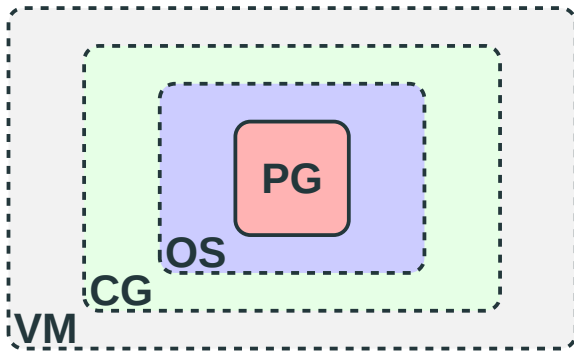


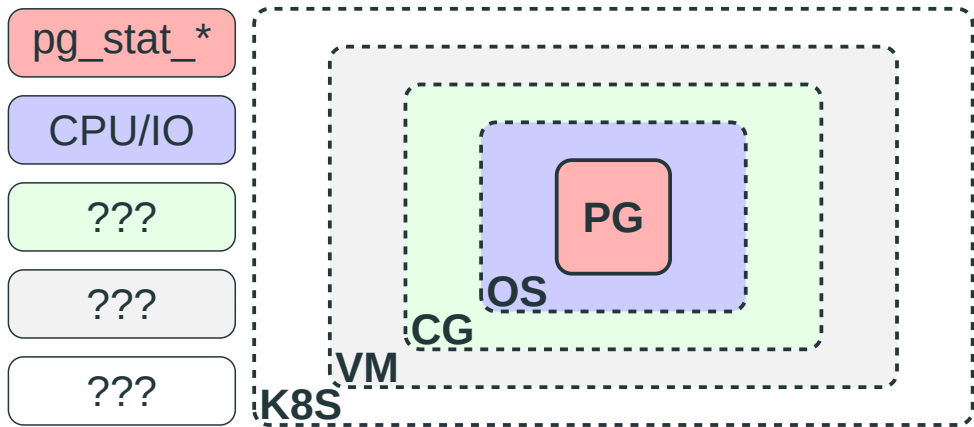
pg\_stat\_\*

CPU/IO

???

???







**All right, then. Keep your secrets**



**Plan?**

# A bit chaotic



## Info sources

source code

strace/GDB/Perf

procfs/sysfs

BPF/eBPF/BCC

## Shared memory

```
ERROR:  could not resize shared memory segment  
"/PostgreSQL.699663942" to 50438144 bytes:  
No space left on device
```

```
# strace -k -p PID
openat(AT_FDCWD, "/dev/shm/PostgreSQL.62223175"
ftruncate(176, 50438144) = 0
fallocate(176, 0, 0, 50438144) = -1 ENOSPC
> libc-2.27.so(posix_fallocate+0x16) [0x114f76]
> postgres(dsm_create+0x67) [0x377067]
...
> postgres(ExecInitParallelPlan+0x360) [0x254a80]
> postgres(ExecGather+0x495) [0x269115]
> postgres(standard_ExecutorRun+0xfd) [0x25099d]
...
> postgres(exec_simple_query+0x19f) [0x39afdf]
```

## vDSO

```
# strace -k -p PID on XEN  
gettimeofday({tv_sec=1550586520, tv_usec=313499}, NULL) = 0  
> [vdso]() [0xef0]
```

Two frequently used system calls are 77% slower on AWS EC2

## # Experiment 1

transaction type: pg\_long.sql

latency average = **1312**.903 ms

## # Experiment 2

SQL script **1**: pg\_long.sql

- weight: **1** (targets **50.0%** of total)

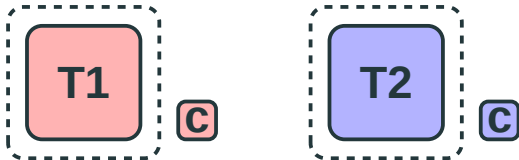
- latency average = **1426**.928 ms

SQL script **2**: pg\_short.sql

- weight: **1** (targets **50.0%** of total)

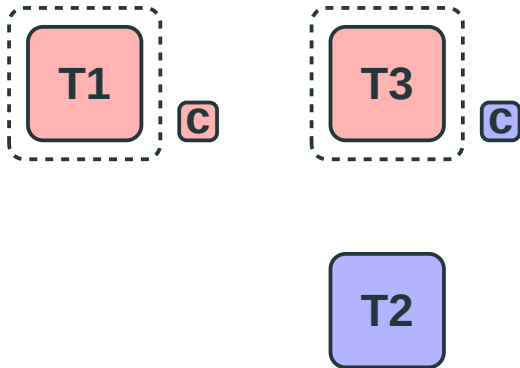
- latency average = **303**.092 ms

# Scheduling





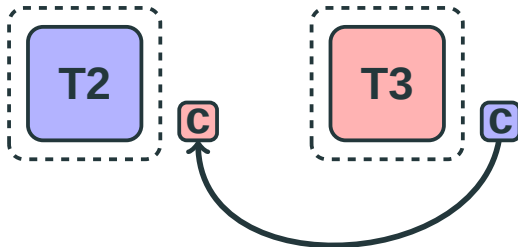
# Scheduling



# Scheduling



# Scheduling



```
# perf record -e cache-misses,cpu-migrations
```

```
# Experiment 1
```

```
12,396,382,649
```

```
2,750
```

```
cache-misses # 28.562%
```

```
cpu-migrations
```

```
# Experiment 2
```

```
20,665,817,234
```

```
10,460
```

```
cache-misses # 28.533%
```

```
cpu-migrations
```

time	cpu	01234	task name [tid/pid]	wait time (msec)	sch delay (msec)	run time (msec)	
4227.834476	[0003]	s	postgres[12935]	0.000	0.000	0.000	
4227.834895	[0003]	s	postgres[12935]	0.000	0.000	0.418	
4227.835478	[0003]	s	postgres[25080]	0.000	0.040	0.583	
4227.836485	[0003]	s	postgres[25080]	0.000	0.000	1.007	
4227.837482	[0003]	s	postgres[25080]	0.000	0.000	0.996	
4227.837784	[0003]	s	postgres[25080]	0.000	0.000	0.302	
4227.837989	[0003]	m	postgres[25080]				migrated: :25077 cpu 1 => 3
4227.837993	[0003]	s	postgres[25080]	0.000	0.000	0.208	
4227.848487	[0003]	s	postgres[25080]	0.000	0.000	10.493	
4227.848991	[0003]	s	postgres[25080]	0.000	0.000	0.504	
4227.849487	[0003]	s	postgres[25080]	0.000	0.000	0.495	
4227.849748	[0003]	s	postgres[25080]	0.000	0.000	0.260	
4227.849912	[0003]	s	postgres[25080]	0.000	0.000	0.164	
4227.851477	[0001]	s	postgres[25082]	0.000	0.000	0.000	
4227.851481	[0002]	s	postgres[25080]	0.000	0.000	0.000	
4227.851778	[0003]	s	postgres[12935]	15.017	0.000	1.866	
4227.852259	[0003]	m	postgres[12935]				migrated: postgres[25083] cpu 1 => 3
4227.852263	[0003]	s	postgres[12935]	0.000	0.000	0.484	
4227.852477	[0003]	s	postgres[25083]	0.000	0.058	0.214	
4227.852478	[0001]	s	postgres[25082]	0.000	0.000	1.001	
4227.852614	[0002]	s	postgres[12935]	0.000	0.000	1.133	

## Huge pages

transparent vs classic

TLB misses are faster and less frequent

# Huge pages

```
# perf record -e dTLB-loads,dTLB-stores -p PID
```

```
# huge_pages on
```

```
Samples: 832K of event 'dTLB-load-misses'
```

```
Event count (approx.): 640614445 : ~19% less
```

```
Samples: 736K of event 'dTLB-store-misses'
```

```
Event count (approx.): 72447300 : ~29% less
```

```
# huge_pages off
```

```
Samples: 894K of event 'dTLB-load-misses'
```

```
Event count (approx.): 784439650
```

```
Samples: 822K of event 'dTLB-store-misses'
```

```
Event count (approx.): 101471557
```

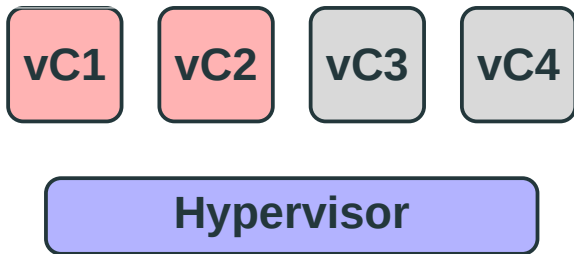
# VM

- Lock holder preemption problem
- Lock waiter preemption problem
- Intel PLE (pause loop exiting)
- PLE\_Gap, PLE\_Window

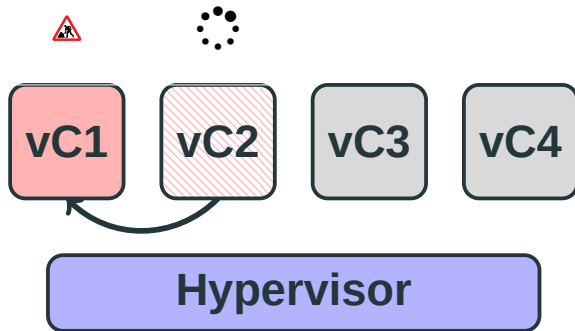
Intel® 64 and IA-32 Architectures Software Developer's Manual, Vol. 3



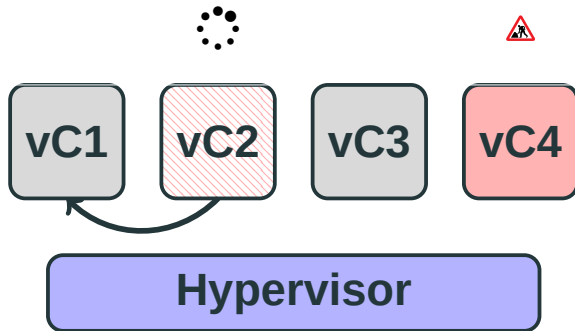
# vCPU



## vCPU

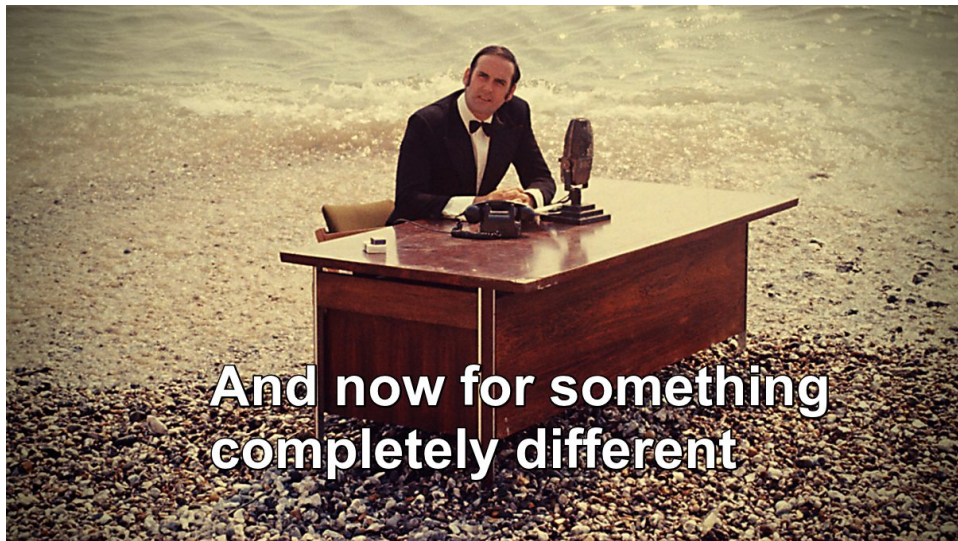


# vCPU



```
# latency average = 17.782 ms  
=> modprobe kvm-intel ple_gap=128  
=> perf record -e kvm:kvm_exit  
reason PAUSE_INSTRUCTION 306795
```

```
# latency average = 16.858 ms  
=> modprobe kvm-intel ple_gap=0  
=> perf record -e kvm:kvm_exit  
reason PAUSE_INSTRUCTION 0
```



**And now for something  
completely different**

## Tunables

```
# from /proc/sys/kernel/  
sched_wakeup_granularity_ns  
# default = 1 msec * (1 + ilog(ncpus))
```

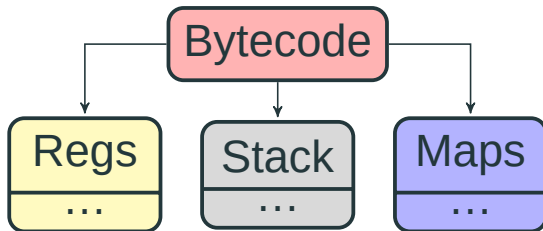
# Userspace

---

Bytecode

# Userspace

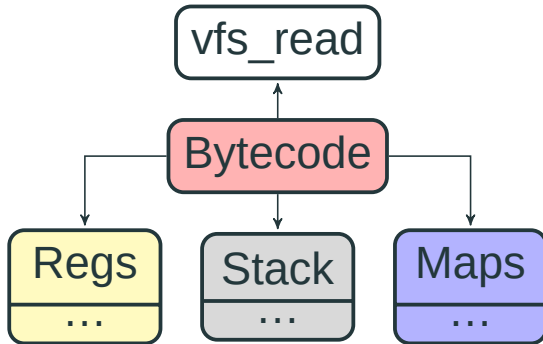
---





# Userspace

---



# pgbench and pg\_dump

real 1m38.990s  
user 1m9.127s  
sys 0m2.066s

usecs	: count	distribution
<b>0 -&gt; 1</b>	<b>: 16</b>	
<b>2 -&gt; 3</b>	<b>: 4604</b>	**
<b>4 -&gt; 7</b>	<b>: 6812</b>	****
<b>8 -&gt; 15</b>	<b>: 14888</b>	*****
<b>16 -&gt; 31</b>	<b>: 19267</b>	*****
<b>32 -&gt; 63</b>	<b>: 65795</b>	*****
<b>64 -&gt; 127</b>	<b>: 50454</b>	*****
<b>128 -&gt; 255</b>	<b>: 16393</b>	*****
<b>256 -&gt; 511</b>	<b>: 5981</b>	***
<b>512 -&gt; 1023</b>	<b>: 12300</b>	*****
<b>1024 -&gt; 2047</b>	<b>: 48</b>	
<b>2048 -&gt; 4095</b>	<b>: 0</b>	

# pgbench and pg\_dump

```
real    1m32.030s
user    1m8.559s
sys     0m1.641s
```

usecs	: count	distribution
<b>0 -&gt; 1</b>	<b>: 1</b>	
<b>2 -&gt; 3</b>	<b>: 8</b>	
<b>4 -&gt; 7</b>	<b>: 25</b>	
<b>8 -&gt; 15</b>	<b>: 46</b>	*
<b>16 -&gt; 31</b>	<b>: 189</b>	*****
<b>32 -&gt; 63</b>	<b>: 119</b>	****
<b>64 -&gt; 127</b>	<b>: 96</b>	***
<b>128 -&gt; 255</b>	<b>: 93</b>	***
<b>256 -&gt; 511</b>	<b>: 238</b>	*****
<b>512 -&gt; 1023</b>	<b>: 323</b>	*****
<b>1024 -&gt; 2047</b>	<b>: 1012</b>	*****
<b>2048 -&gt; 4095</b>	<b>: 47</b>	*

[github.com/iovisor/bcc/](https://github.com/iovisor/bcc/)

[github.com/erthalion/postgres-bcc](https://github.com/erthalion/postgres-bcc)

# Cache

Cache contention

Intel RDT

Class of service

# Cache

=> llcache\_per\_query.py bin/postgres

PID	QUERY	CPU	REFERENCE	MISS	HIT%
<b>9720</b>	UPDATE pgbench_tellers ...	<b>0</b>	<b>2000</b>	<b>1000</b>	<b>50.00%</b>
<b>9720</b>	SELECT abalance FROM ...	<b>2</b>	<b>2000</b>	<b>100</b>	<b>95.00%</b>
...					

Total References: **3303100** Total Misses: **599100** Hit Rate: **81.86%**

## Shared memory

```
ERROR: could not resize shared memory segment  
"/PostgreSQL.699663942" to 50483144 bytes:  
No space left on device
```

5

Remember?



# Shared memory

=> shmem.py bin/postgres

mmap:

**[20439]**: 142M

anon shm:

**[20439]**: 56B

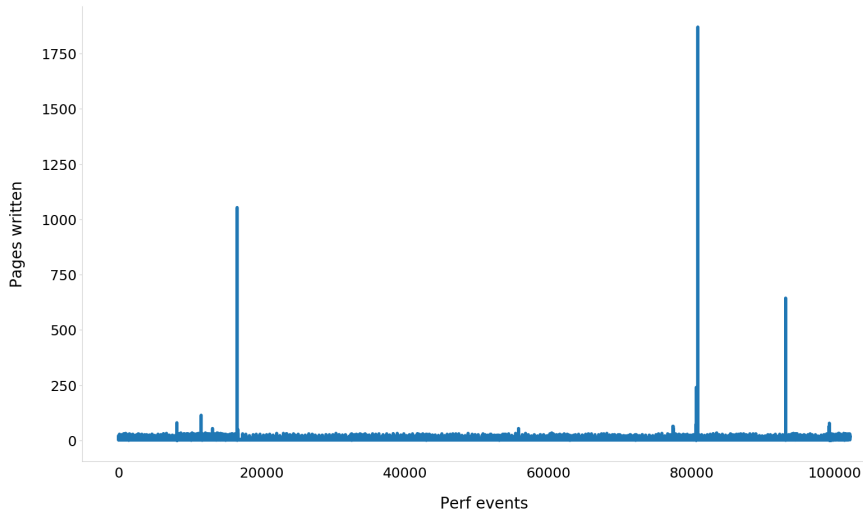
shm:

**[postmaster.opts]**: 0B

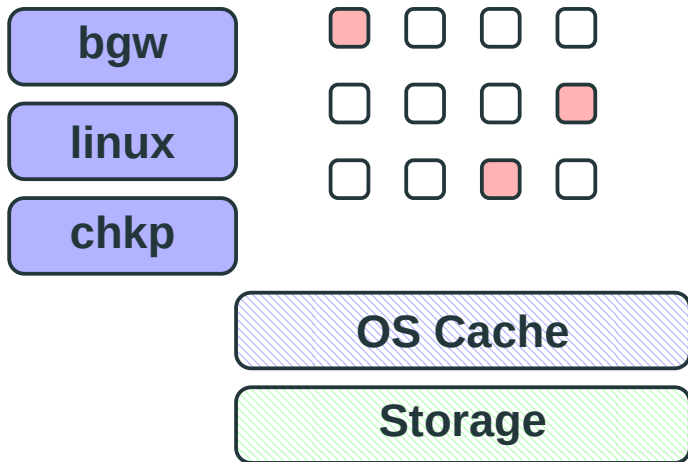
**[PostgreSQL.57332071]**: 7K



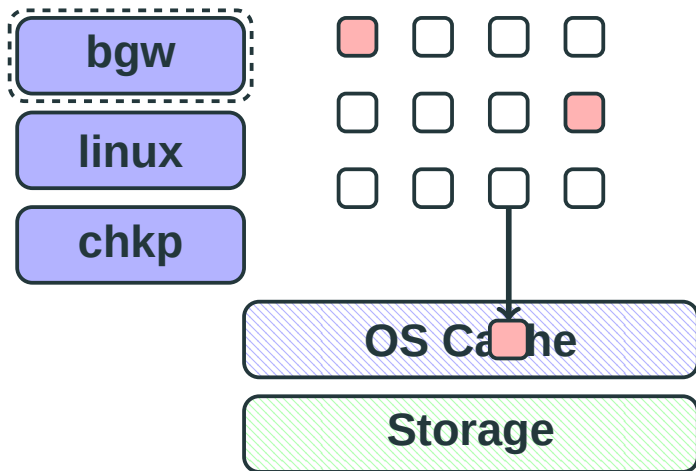
# Pages written, kernel



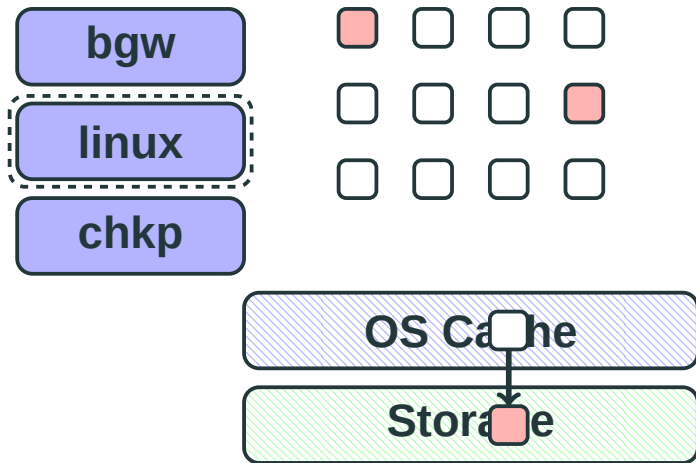
## Dirty pages



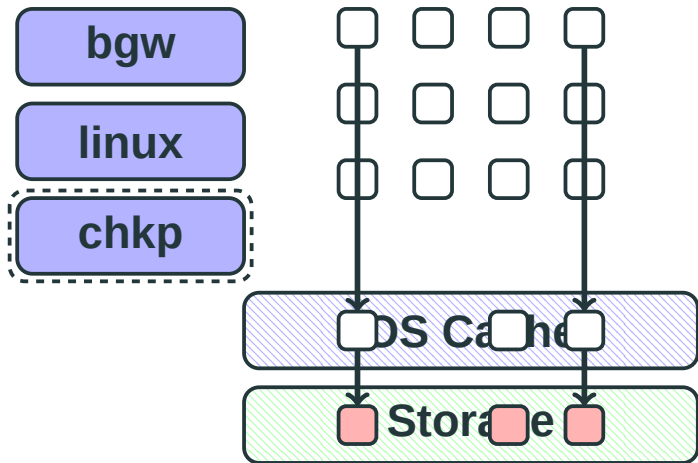
# Dirty pages



# Dirty pages



# Dirty pages



## Writeback (cgroup v1)

```
/* vmscan.c */  
/* The normal page dirty throttling mechanism  
 * in balance_dirty_pages() is completely broken  
 * with the legacy memcg and direct stalling in  
 * shrink_page_list() is used for throttling instead,  
 * which lacks all the niceties such as fairness,  
 * adaptive pausing, bandwidth proportional  
 * allocation and configurability.  
 */  
static bool sane_reclaim(struct scan_control *sc)
```

# Writeback

```
=> perf record -e writeback:writeback_written
```

```
kworker/u8:1 reason=periodic    nr_pages=101429  
kworker/u8:1 reason=background  nr_pages=MAX_ULONG  
kworker/u8:3 reason=periodic    nr_pages=101457
```

# Writeback

```
# pgbench insert workload
```

```
=> io_timeouts.py bin/postgres
```

```
[18335] END: MAX_SCHEDULE_TIMEOUT
```

```
[18333] END: MAX_SCHEDULE_TIMEOUT
```

```
[18331] END: MAX_SCHEDULE_TIMEOUT
```

```
[18318] truncate pgbench_history: MAX_SCHEDULE_TIMEOUT
```



# Kubernetes

```
resources:  
  requests:  
    memory: "64Mi"  
    cpu: "250m"  
  limits:  
    memory: "128Mi"  
    cpu: "500m"
```

# Kubernetes

resources:

requests:

memory: "64Mi"

cpu: "250m"

limits:

memory: "128Mi"

cpu: "500m"

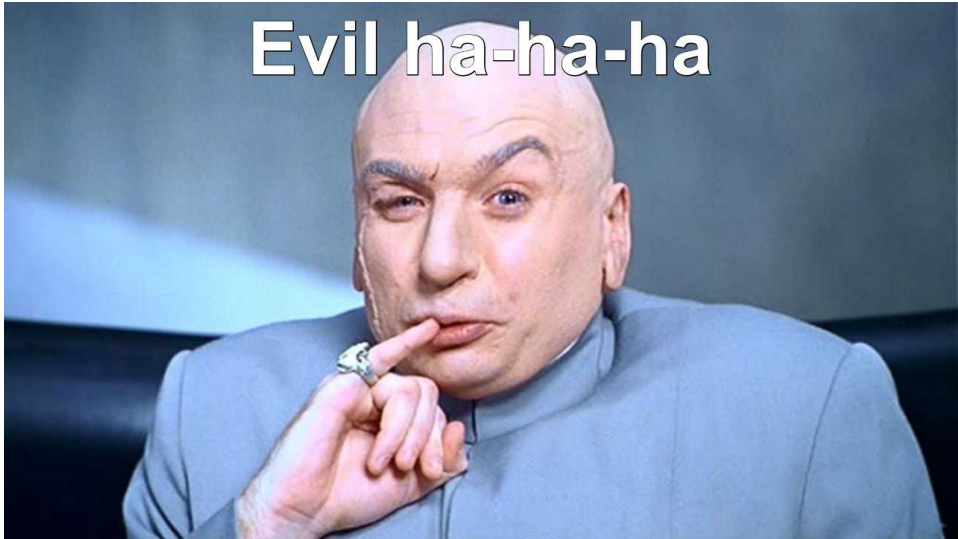


soft\_limits\_in\_bytes



limits\_in\_bytes

Evil ha-ha-ha



# Kubernetes

resources:

requests:

memory: "64Mi"

cpu: "250m"

limits:

memory: "128Mi"

cpu: "500m"

~~soft\_limits\_in\_bytes~~

limits\_in\_bytes

# Memory reclaim

# only under the memory pressure

=> `page_reclaim.py --container 89c33bb3133f`

**[7382]** postgres: 928K

**[7138]** postgres: 152K

**[7136]** postgres: 180K

**[7468]** postgres: 72M

**[7464]** postgres: 57M

**[5451]** postgres: 1M

## How to run?

```
# bcc + postgres-bcc
```

```
CONFIG_BPF=y
```

```
CONFIG_BPF_SYSCALL=y
```

```
CONFIG_NET_CLS_BPF=m
```

```
CONFIG_NET_ACT_BPF=m
```

```
CONFIG_BPF_JIT=y
```

```
CONFIG_BPF_EVENTS=y
```

```
debugfs on /sys/kernel/debug type debugfs (rw)
```

## How to run: container?

```
# sometimes you also need to let perf know  
# where to find debugging symbols, e.g. copy  
# from /usr/lib/.debug/  
docker run  
    --privileged  
    --net=container:<container-id>  
    --ipc=container:<container-id>
```

## How to run: K8S?

spec:

serviceAccountName: "bcc"

hostPID: true

containers:

- name: "bcc"

securityContext:

privileged: true

# 4 \* 65536 + 14 \* 256 + 96

=> export BCC\_LINUX\_VERSION\_CODE **265824**



## How to break?

```
# unsafe access
```

```
=> perf probe -x bin/postgres --funcs
```

```
=> perf probe -x bin/postgres 'ExecCallTriggerFunc trigdata->?'
```

```
=> perf record probe_postgres:ExecCallTriggerFunc
```

## How to break?

# non interruptible sleep

=> perf probe -x bin/postgres --funcs

=> perf probe -x bin/postgres 'XLogInsertRecord fpw\_lsn'

# How to break?

iovisor / bcc

Watch 410 ★ Uns

<> Code ⓘ Issues 266 🔗 Pull requests 21 📁 Projects 0 📖 Wiki 📊 Insights

## Ubuntu xenial kernel panic in bpf\_map\_update\_elem using ext4slower #1678


**Closed** stefreak opened this issue on Apr 12, 2018 · 13 comments


## Questions?

 [github.com/erthalion](https://github.com/erthalion)

 [github.com/erthalion/postgres-bcc](https://github.com/erthalion/postgres-bcc)

 [@erthalion](https://twitter.com/erthalion)

 [dmitrii.dolgov at zalando dot de](mailto:dmitrii.dolgov@zalando.de)

 [9erthalion6 at gmail dot com](mailto:9erthalion6@gmail.com)