Improving the Performance of your Kubernetes Cluster

Priya Wadhwa

About me

- Maintainer of minikube
- Maintained open source projects including skaffold and kaniko in the past
- Complete beginner to performance engineering (as of 8 months ago)

priyawadhwa@

priyawadhwa16@

About my machine

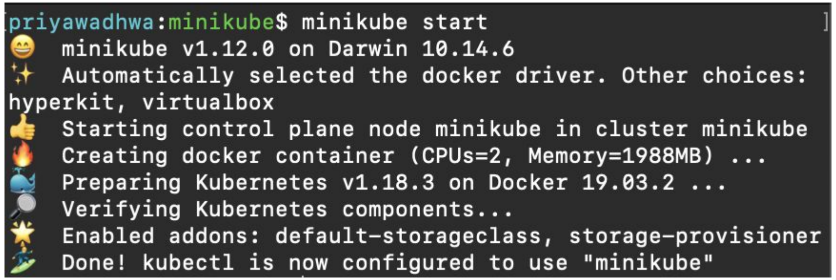
```
$ system_profiler SPHardwareDataType
Hardware:
    Hardware Overview:
      Model Name: MacBook Pro
      Model Identifier: MacBookPro15,3
      Processor Name: Intel Core i9
      Processor Speed: 2.3 GHz
      Number of Processors: 1
      Total Number of Cores: 8
      L2 Cache (per Core): 256 KB
      L3 Cache: 16 MB
      Hyper-Threading Technology: Enabled
      Memory: 32 GB
```

Agenda

- How I learned to use performance tools (Linux & Go)
- How I used analyses from these tools to improve k8s overhead
- Case study in minikube, a local k8s cluster

What's minikube?

- Run a kubernetes cluster locally
- Runs either in a VM or as a container in Docker
- Easy way to get started with k8s!
- Typically users run a local single node cluster

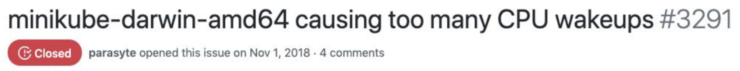


```
priyawadhwa:minikube$ minikube start
😄  minikube v1.12.0 on Darwin 10.14.6
✨  Automatically selected the docker driver. Other choices:
hyperkit, virtualbox
👍  Starting control plane node minikube in cluster minikube
🔥  Creating docker container (CPUs=2, Memory=1988MB) ...
🐳  Preparing Kubernetes v1.18.3 on Docker 19.03.2 ...
🔎  Verifying Kubernetes components...
🌟  Enabled addons: default-storageclass, storage-provisioner
🏄  Done! kubectl is now configured to use "minikube"
```

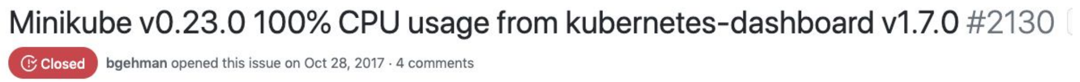
github.com/kubernetes/minikube

minikube roundtable

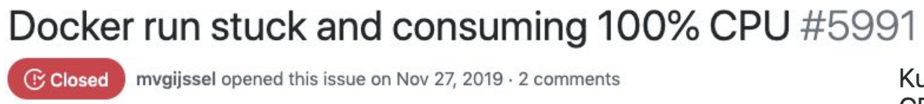
"Burning the legs off of developers since 2016"


minikube-darwin-amd64 causing too many CPU wakeups #3291

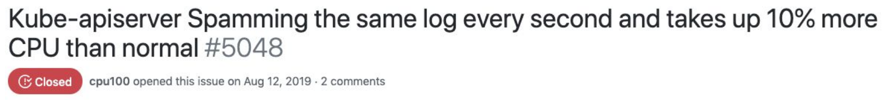
Closed parasyte opened this issue on Nov 1, 2018 · 4 comments


Minikube v0.23.0 100% CPU usage from kubernetes-dashboard v1.7.0 #2130

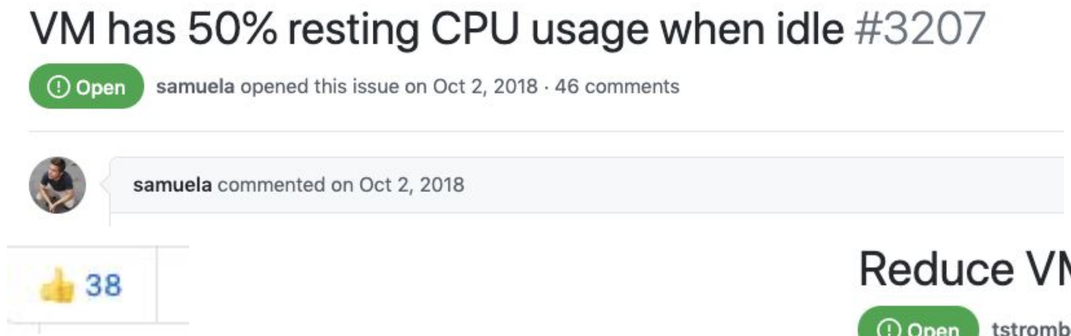
Closed bgehman opened this issue on Oct 28, 2017 · 4 comments


Docker run stuck and consuming 100% CPU #5991

Closed mvgijssel opened this issue on Nov 27, 2019 · 2 comments


Kube-apiserver Spamming the same log every second and takes up 10% more CPU than normal #5048

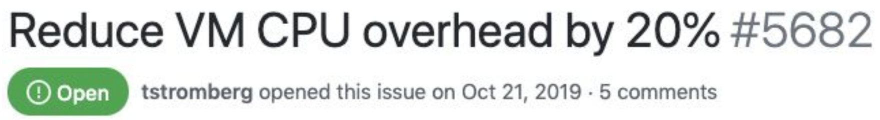
Closed cpu100 opened this issue on Aug 12, 2019 · 2 comments


VM has 50% resting CPU usage when idle #3207

Open samuela opened this issue on Oct 2, 2018 · 46 comments

samuela commented on Oct 2, 2018

👍 38


Reduce VM CPU overhead by 20% #5682

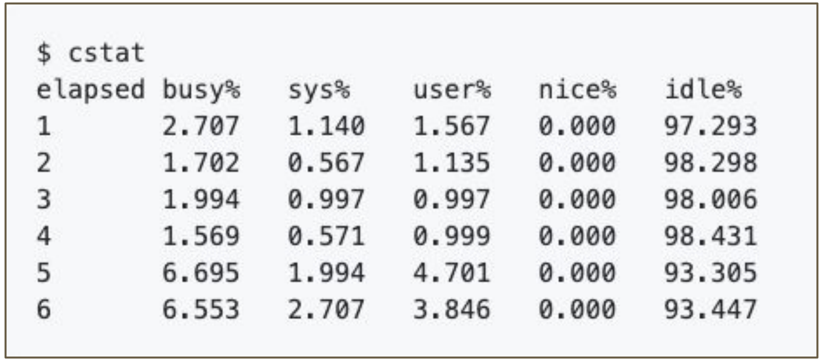
Open tstromberg opened this issue on Oct 21, 2019 · 5 comments

github.com/kubernetes/minikube

THIS IS FINE.

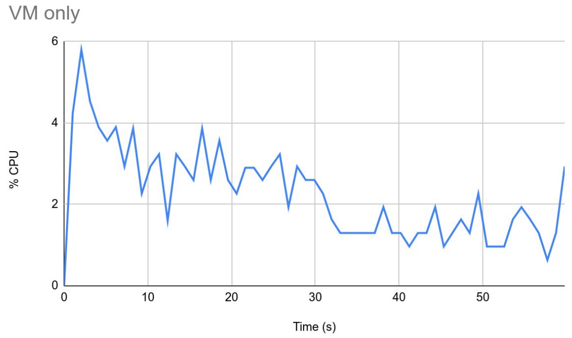
Step 1: Calculating overhead

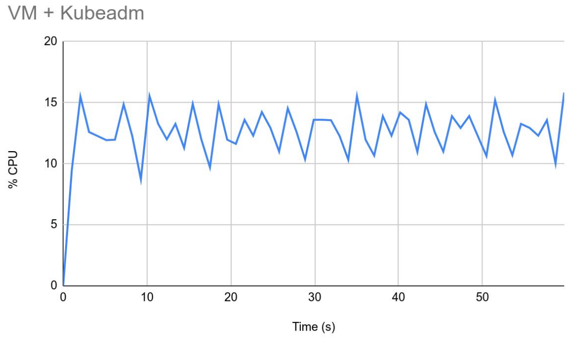
- Calculating overhead of *a single process*
  - github.com/priyawadhwa/track-cpu
  - The same as running ps
- Calculating overhead of *the entire system*
  - github.com/tstromberg/cstat
  - More precise `iostat`
  - (system overhead with minikube) - (system overhead without minikube)

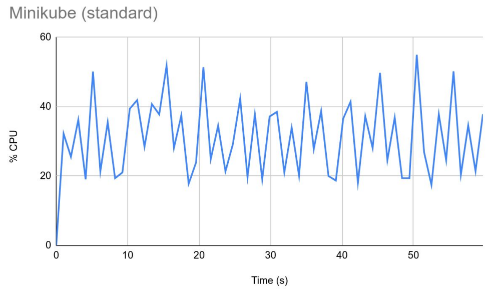


```
$ cstat
elapsed busy%   sys%    user%   nice%   idle%
1       2.707   1.140   1.567   0.000   97.293
2       1.702   0.567   1.135   0.000   98.298
3       1.994   0.997   0.997   0.000   98.006
4       1.569   0.571   0.999   0.000   98.431
5       6.695   1.994   4.701   0.000   93.305
6       6.553   2.707   3.846   0.000   93.447
```

Where is the overhead coming from?







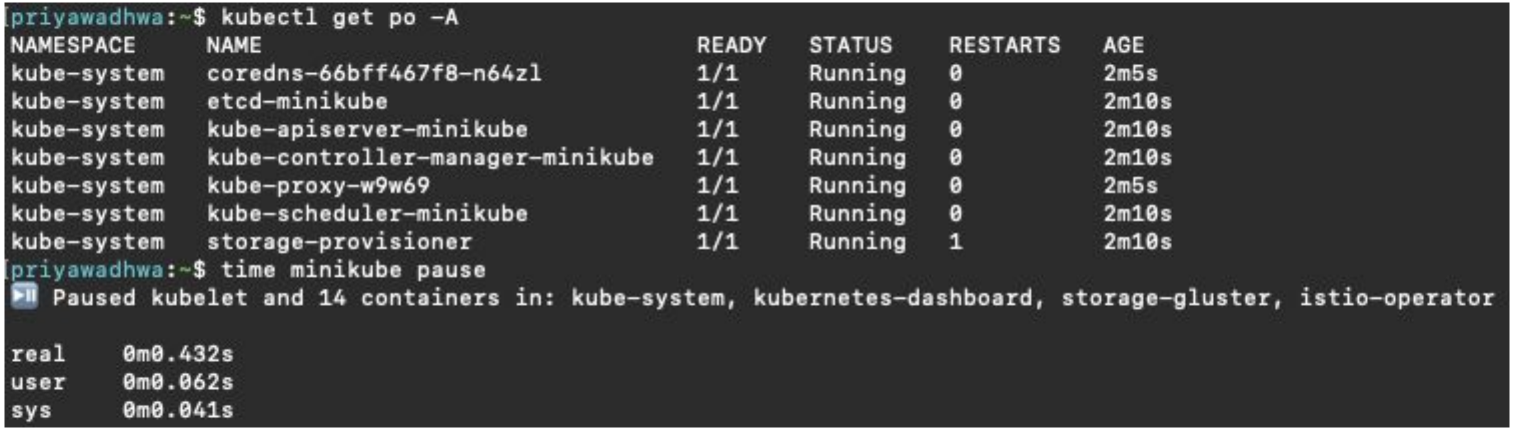
0-4%

10-15%

20-40%

minikube pause/unpause

- Now we know that running a VM is inexpensive
- Inspired the minikube pause command, which stops all kubernetes containers in the VM
- Runs user’s application without the overhead of k8s
- Takes <1s to pause/unpause a cluster



```
priyawadhwa:~$ kubectl get po -A
NAMESPACE     NAME                               READY   STATUS    RESTARTS   AGE
kube-system   coredns-66bff467f8-n64zl           1/1     Running   0          2m5s
kube-system   etcd-minikube                      1/1     Running   0          2m10s
kube-system   kube-apiserver-minikube            1/1     Running   0          2m10s
kube-system   kube-controller-manager-minikube   1/1     Running   0          2m10s
kube-system   kube-proxy-w9w69                   1/1     Running   0          2m5s
kube-system   kube-scheduler-minikube            1/1     Running   0          2m10s
kube-system   storage-provisioner                1/1     Running   1          2m10s
priyawadhwa:~$ time minikube pause
⏯  Paused kubelet and 14 containers in: kube-system, kubernetes-dashboard, storage-gluster, istio-operator

real	0m0.432s
user	0m0.062s
sys	0m0.041s
```

How do we improve the performance of our kubernetes cluster?

Learning to use performance tools

Linux performance tools:

- Linux perf_events
- Flamegraphs
- eBPF tools

Go performance tools:

- pprof

Linux performance tools

- The USE method by Brendan Gregg
  - Utilization, Saturation and Errors
  - www.brendangregg.com/usemethod.html
  - www.brendangregg.com/USEmethod/use-linux.html
- eBPF Tools
  - http://www.brendangregg.com/ebpf.html
- Flame graphs
  - http://www.brendangregg.com/flamegraphs.html

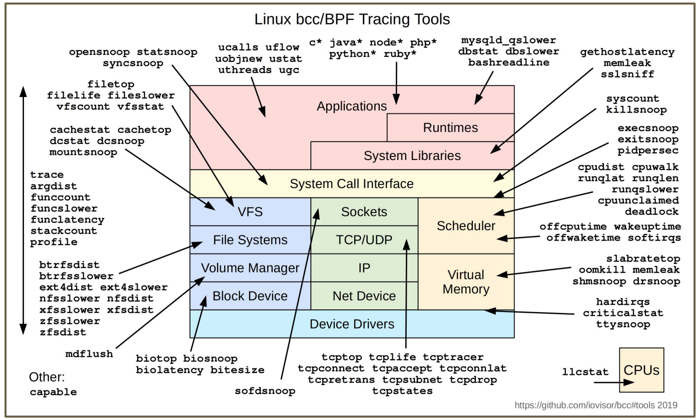
The USE Method

- The USE method by Brendan Gregg
    - Utilization, Saturation and Errors
    - www.brendangregg.com/usemethod.html
    - www.brendangregg.com/USEmethod/use-linux.html

Component	Command	Conclusion
CPU (system wide)	mpstat -P ALL 1	System CPU normal, no single core is being overloaded
CPU (minikube process)	pidstat 1 -C qemu --human (for 11 seconds)	Average 22% overhead of minikube process, 14.5% coming from within the VM
Memory capacity	free -m	Normal
Storage Device I/O (Utilization)	iostat -xz 1	Normal
Storage Device I/O (Utilization, per process)	sudo iotop --only	Minikube has multiple process writing to disk at once

eBPF in Minikube

- eBPF = Extended Berkeley Packet Filter
- Recommended front ends are BCC tools
    - Huge collection of tracing tools in Python
    - These tools profile and trace the Linux kernel
    - Instructions for running bcc tools in minikube can be found at https://minikube.sigs.k8s.io/docs/tutorials/ebpf_tools_in_minikube/


Linux bcc/BPF Tracing Tools
opensnoop statsnoop syncsnoop
ucalls uflow uobjnew ustat uthreads ugc
c* java* node* php* python* ruby*
mysqld_qslower dbstat dbslower bashreadline
gethostlatency memleak sslsniff
filetop filelife fileslower vfscount vfsstat
cachestat cachetop dcstat dcsnoop mountsnoop
trace argdist funccount funcslower funclatency stackcount profile
btrfsdist btrfsslower ext4dist ext4slower nfsslower nfsdist xfsslower xfsdist zfsslower zfsdist
mdflush
Applications
Runtimes
System Libraries
System Call Interface
VFS
Sockets
Scheduler
File Systems
TCP/UDP
Volume Manager
IP
Virtual Memory
Block Device
Net Device
Device Drivers
syscount killsnoop
execsnoop exitsnoop pidpersec
cpudist cpuwalk runqlat runqlen runqslower cpuunclaimed deadlock
offcputime wakeuptime offwaketime softirqs
slabratetop oomkill memleak shmsnoop drsnoop
hardirqs criticalstat ttysnoop
biotop biosnoop biolatency bitesize
sofdsnoop
tcptop tcplife tcptracer tcpconnect tcpaccept tcpconnlat tcpretrans tcpsubnet tcpdrop tcpstates
llcstat
CPUs
Other: capable
https://github.com/iovisor/bcc#tools 2019

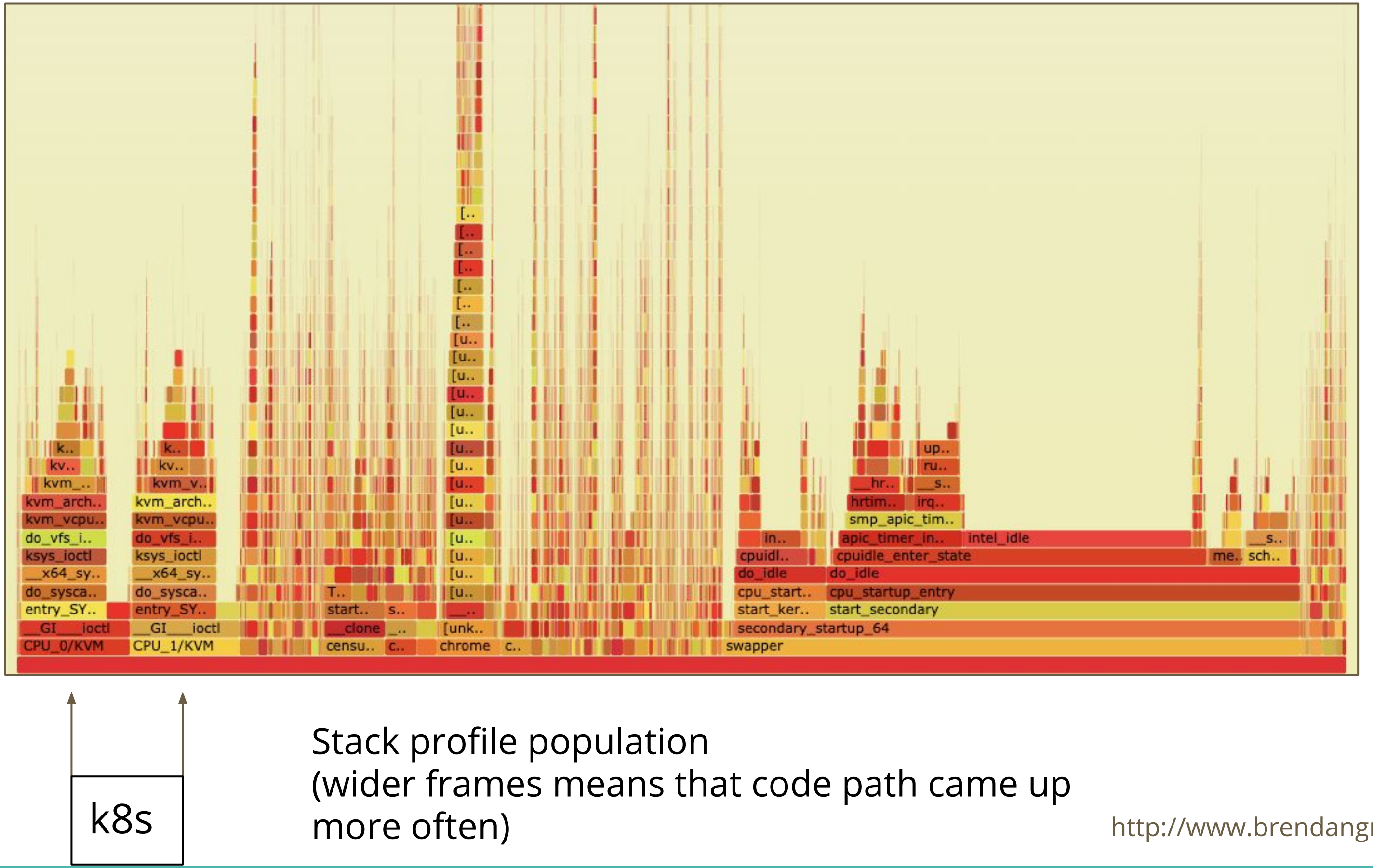
https://github.com/iovisor/bcc

http://www.brendangregg.com/ebpf.html

biosnoop - Trace block device I/O with PID and latency

```
$ minikube ssh -- docker run --rm   --privileged   -v /lib/modules:/lib/modules:ro   -v /usr/src:/usr/src:ro   -v
/etc/localtime:/etc/localtime:ro   --workdir /usr/share/bcc/tools   zlim/bcc ./biosnoop

^CTIME(s)       COMM           PID    DISK    T  SECTOR    BYTES    LAT(ms)
0.000000000    etcd           3466   vda     W  2754384   4096      0.33
0.018458000    etcd           3466   vda     W  3010584   4096      0.25
0.020495000    etcd           3466   vda     W  17148296  4096      0.19
3.077617000    etcd           3707   vda     W  2754384   4096      0.28
3.129560000    etcd           3707   vda     W  3010576   4096      0.23
3.129585000    etcd           3707   vda     W  3010592   12288     0.23
3.131553000    etcd           3707   vda     W  17148304  4096      0.15
4.918491000    etcd           3707   vda     W  2754384   4096      0.18
4.918521000    jbd2/vda1-8    1840   vda     W  19202488  40960     0.34
4.922187000    jbd2/vda1-8    1840   vda     W  19202568  4096      0.22
4.922503000    etcd           3707   vda     W  2754384   4096      0.18
4.937820000    etcd           3466   vda     W  3010528   4096      0.24
4.937828000    etcd           3466   vda     W  3010408   4096      0.25
4.937832000    etcd           3466   vda     W  3010392   4096      0.27
4.937837000    etcd           3466   vda     W  3010584   4096      0.25
4.939771000    etcd           3466   vda     W  17148296  4096      0.17
5.676038000    etcd           3468   vda     W  2754384   4096      0.27
```

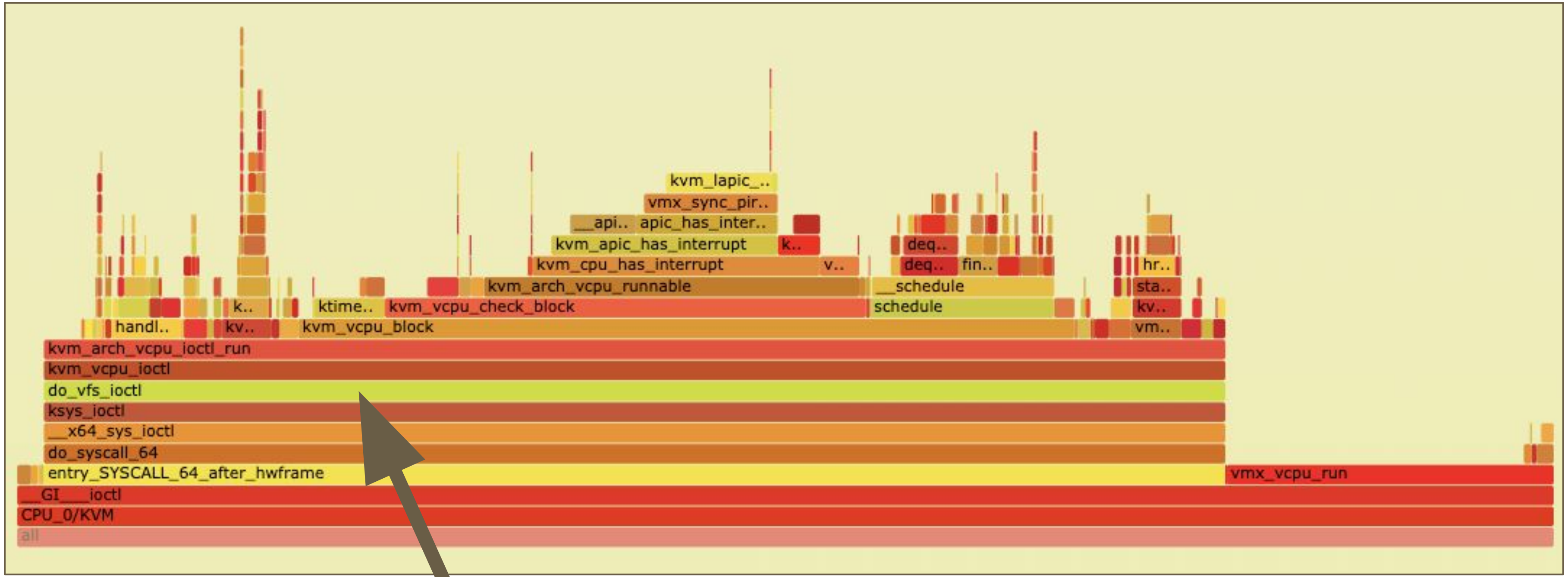
Flame graph



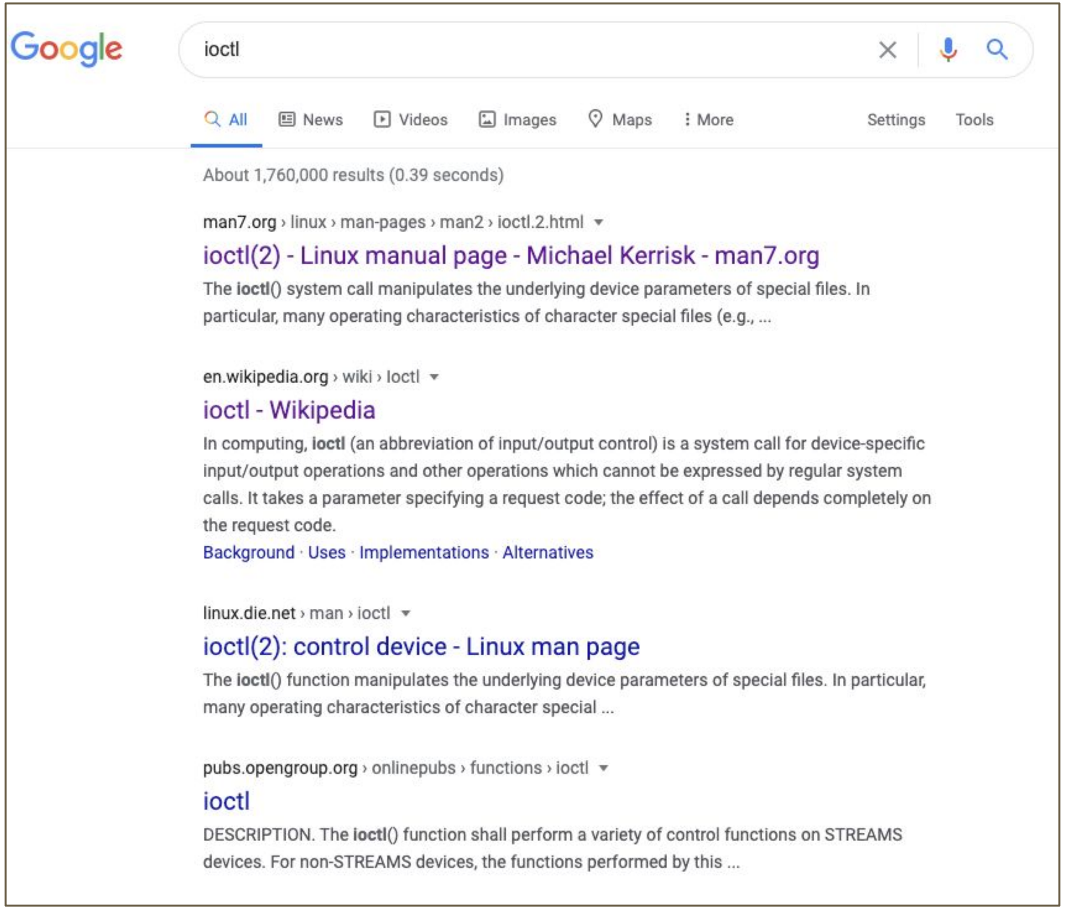
Stack profile population
(wider frames means that code path came up more often)

http://www.brendangregg.com/flamegraphs.html

KVM flamegraph



ioctl?



About 1,760,000 results (0.39 seconds)

man7.org › linux › man-pages › man2 › ioctl.2.html

ioctl(2) - Linux manual page - Michael Kerrisk - man7.org

The **ioctl**() system call manipulates the underlying device parameters of special files. In particular, many operating characteristics of character special files (e.g., ...

en.wikipedia.org › wiki › Ioctl

ioctl - Wikipedia

In computing, **ioctl** (an abbreviation of input/output control) is a system call for device-specific input/output operations and other operations which cannot be expressed by regular system calls. It takes a parameter specifying a request code; the effect of a call depends completely on the request code.

Background · Uses · Implementations · Alternatives

linux.die.net › man › ioctl

ioctl(2): control device - Linux man page

The **ioctl**() function manipulates the underlying device parameters of special files. In particular, many operating characteristics of character special ...

pubs.opengroup.org › onlinepubs › functions › ioctl

ioctl

DESCRIPTION. The **ioctl**() function shall perform a variety of control functions on STREAMS devices. For non-STREAMS devices, the functions performed by this ...

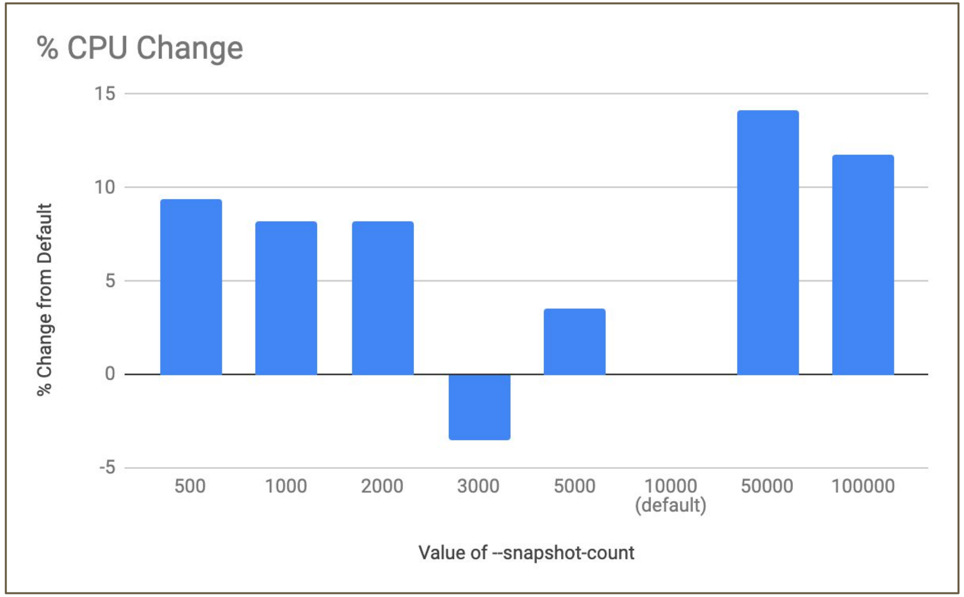
- From my USE analysis, I remembered that etcd was writing to disk a lot
- Maybe this was the cause of the ioctl calls in the flame graph?

```
$ minikube ssh
$ sudo iotop --only

Total DISK READ :       0.00 B/s | Total DISK WRITE :      23.46 K/s
Actual DISK READ:       0.00 B/s | Actual DISK WRITE:      23.46 K/s
  TID  PRIO  USER     DISK READ  DISK WRITE  SWAPIN     IO    COMMAND
 3717 be/4 root        0.00 B/s    3.91 K/s  0.00 %  0.73 % etcd
 3476 be/4 root        0.00 B/s   19.55 K/s  0.00 %  0.33 % etcd
```

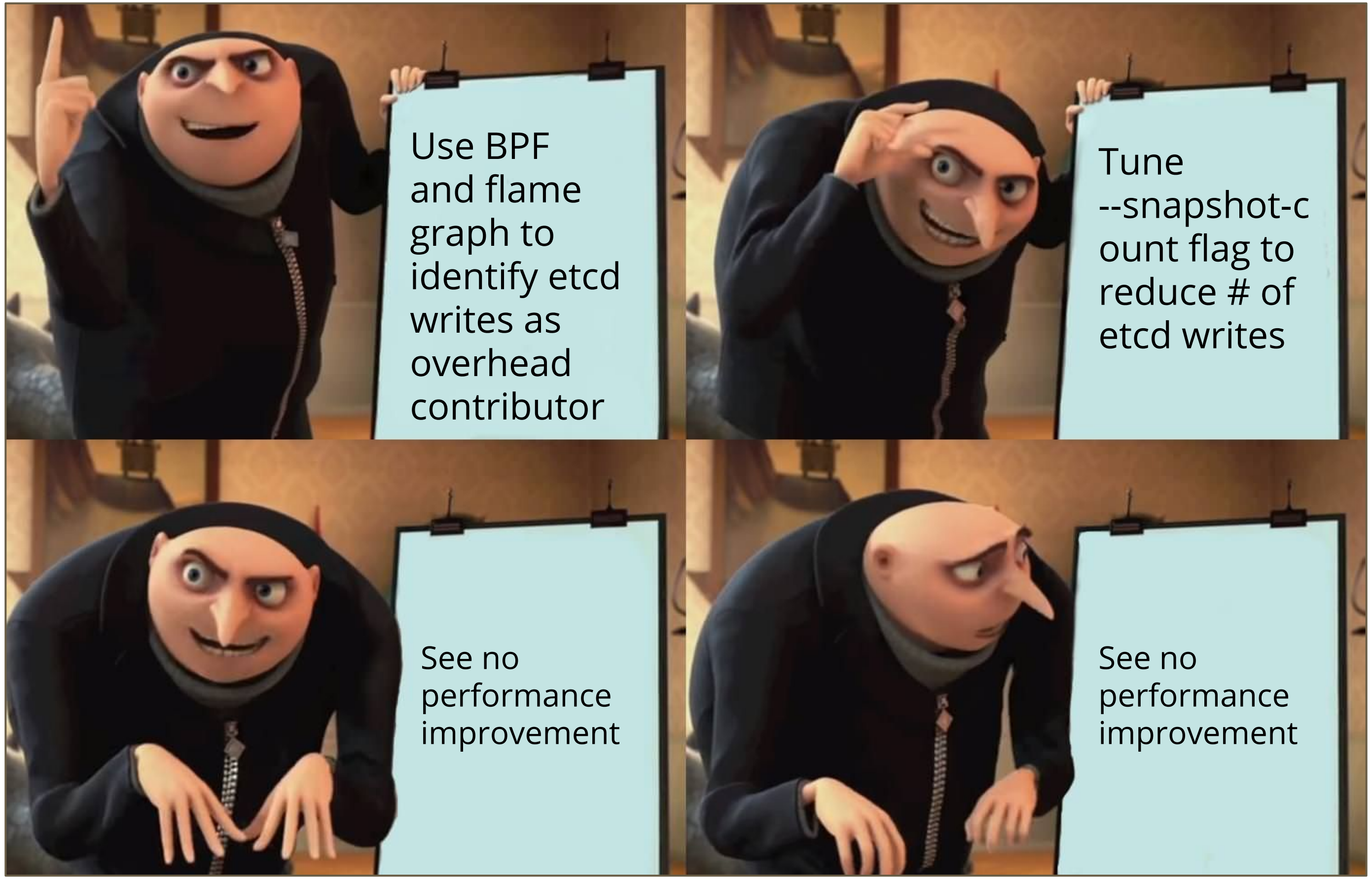
Is there a way to tune how often etcd writes to disk?

`--snapshot-count`: number of committed transactions to trigger a snapshot to disk (default 10,000)

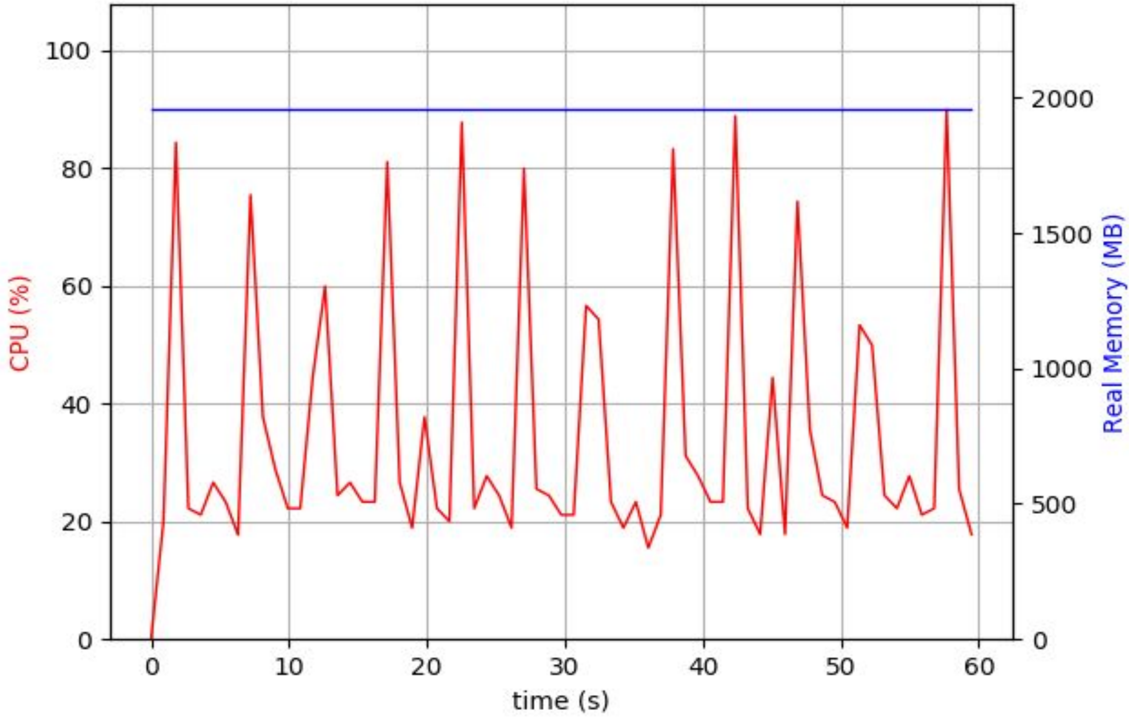
- Tried tuning this value
- Unfortunately, the only improvement was snapshot-count=3000, which only resulted in a 2% improvement (not significant)



Quick Summary


Use BPF and flame graph to identify etcd writes as overhead contributor
Tune --snapshot-count flag to reduce # of etcd writes
See no performance improvement
See no performance improvement

What are these spikes?



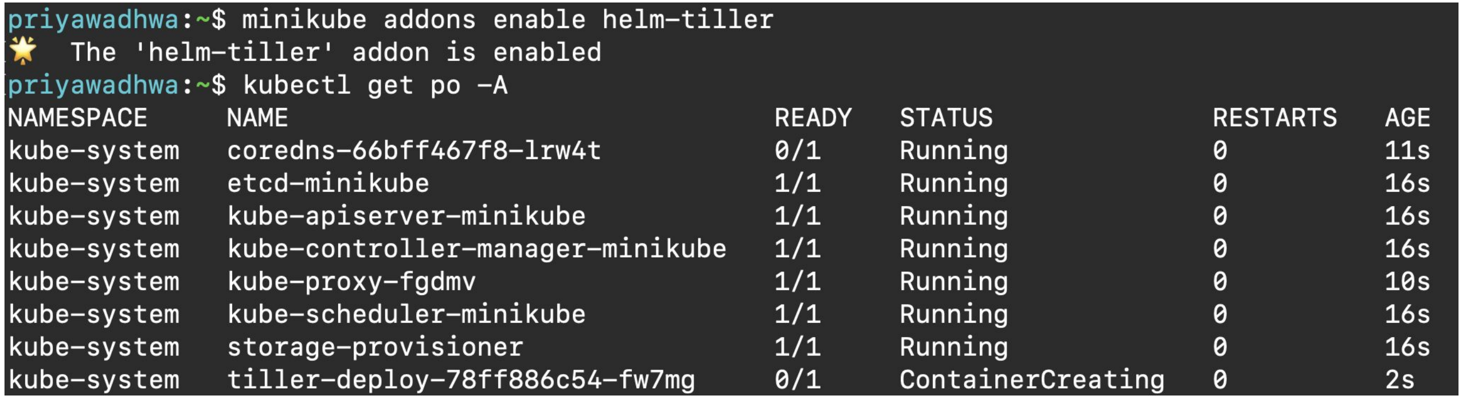
pidstat 1 60

```
15:43:03      UID       PID    %usr %system  %guest   %wait    %CPU   CPU  Command
15:43:04        0      1985    1.00    0.00    0.00    0.00    1.00     1  dockerd
15:43:04        0      3833    3.00    1.00    0.00    0.00    4.00     1  kube-apiserver
15:43:04        0      3870    1.00    0.00    0.00    0.00    1.00     1  kube-controller
15:43:04        0      3896    0.00    1.00    0.00    0.00    1.00     0  etcd
15:43:04        0      3920    1.00    0.00    0.00    0.00    1.00     1  kube-scheduler
15:43:04        0      4323    2.00    1.00    0.00    0.00    3.00     0  kubelet
15:43:04        0      6397    7.00    2.00    0.00    0.00    9.00     0  kubectl

15:43:04      UID       PID    %usr %system  %guest   %wait    %CPU   CPU  Command
15:43:05        0      1985    1.00    1.00    0.00    0.00    2.00     0  dockerd
15:43:05        0      1993    0.00    1.00    0.00    0.00    1.00     1  containerd
15:43:05        0      3833    3.00    1.00    0.00    0.00    4.00     1  kube-apiserver
15:43:05        0      3896    2.00    1.00    0.00    0.00    3.00     0  etcd
15:43:05        0      4323    0.00    1.00    0.00    0.00    1.00     0  kubelet
15:43:05     1000      5660    1.00    0.00    0.00    0.00    1.00     0  sshd
15:43:05        0      6397    3.00    1.00    0.00    0.00    4.00     1  kubectl

15:43:05      UID       PID    %usr %system  %guest   %wait    %CPU   CPU  Command
15:43:06        0      3833    2.00    1.00    0.00    0.00    3.00     1  kube-apiserver
15:43:06        0      3870    0.00    1.00    0.00    0.00    1.00     0  kube-controller
15:43:06        0      3896    0.00    1.00    0.00    0.00    1.00     0  etcd
15:43:06        0      4323    3.00    0.00    0.00    0.00    3.00     0  kubelet
15:43:06        0      5044    0.00    1.00    0.00    0.00    1.00     0  coredns
15:43:06     1000      5660    0.00    1.00    0.00    0.00    1.00     0  sshd
15:43:06     1000      6347    0.00    1.00    0.00    0.00    1.00     1  pidstat
```

```
$ pidstat 1 60 -l

Average:        0     20417    0.60    0.20    0.00    0.00    6.00     -  /usr/local/bin/kubectl
apply -f /etc/kubernetes/addons -l
kubernetes.io/cluster-service!=true,addonmanager.kubernetes.io/mode=Reconcile --prune=true
--prune-whitelist core/v1/ConfigMap --prune-whitelist core/v1/Endpoints --prune-whitelist
core/v1/Namespace --prune-whitelist core/v1/PersistentVolumeClaim --prune-whitelist
core/v1/PersistentVolume --prune-whitelist core/v1/Pod --prune-whitelist
core/v1/ReplicationController --prune-whitelist core/v1/Secret --prune-whitelist core/v1/Service
--prune-whitelist batch/v1/Job --prune-whitelist batch/v1beta1/CronJob --prune-whitelist
apps/v1/DaemonSet --prune-whitelist apps/v1/Deployment --prune-whitelist apps/v1/ReplicaSet
--prune-whitelist apps/v1/StatefulSet --prune-whitelist extensions/v1beta1/Ingress --recursive
```

What is the addon manager?

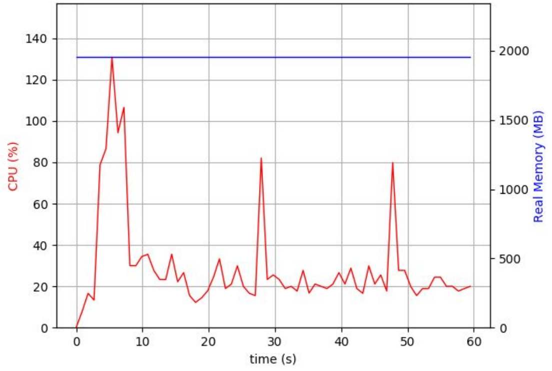
- minikube uses `kube-addon-manager` to enable/disable addons in the cluster
- The addon manager runs `kubectl apply` every 5 seconds to ensure that desired state matches current state



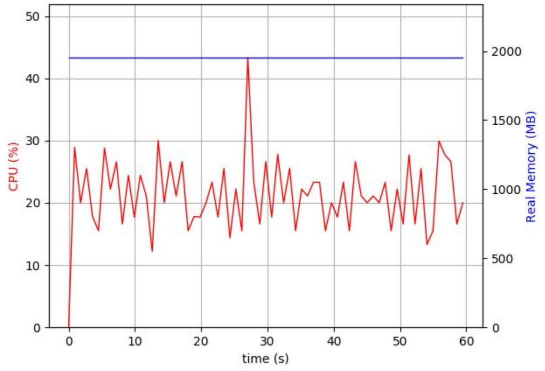
```
priyawadhwa:~$ minikube addons enable helm-tiller
🌟  The 'helm-tiller' addon is enabled
priyawadhwa:~$ kubectl get po -A
NAMESPACE     NAME                                 READY   STATUS              RESTARTS   AGE
kube-system   coredns-66bff467f8-lrw4t             0/1     Running             0          11s
kube-system   etcd-minikube                        1/1     Running             0          16s
kube-system   kube-apiserver-minikube              1/1     Running             0          16s
kube-system   kube-controller-manager-minikube     1/1     Running             0          16s
kube-system   kube-proxy-fgdmv                     1/1     Running             0          10s
kube-system   kube-scheduler-minikube              1/1     Running             0          16s
kube-system   storage-provisioner                  1/1     Running             0          16s
kube-system   tiller-deploy-78ff886c54-fw7mg       0/1     ContainerCreating   0          2s
```

Hypothesis: it's the addon manager!

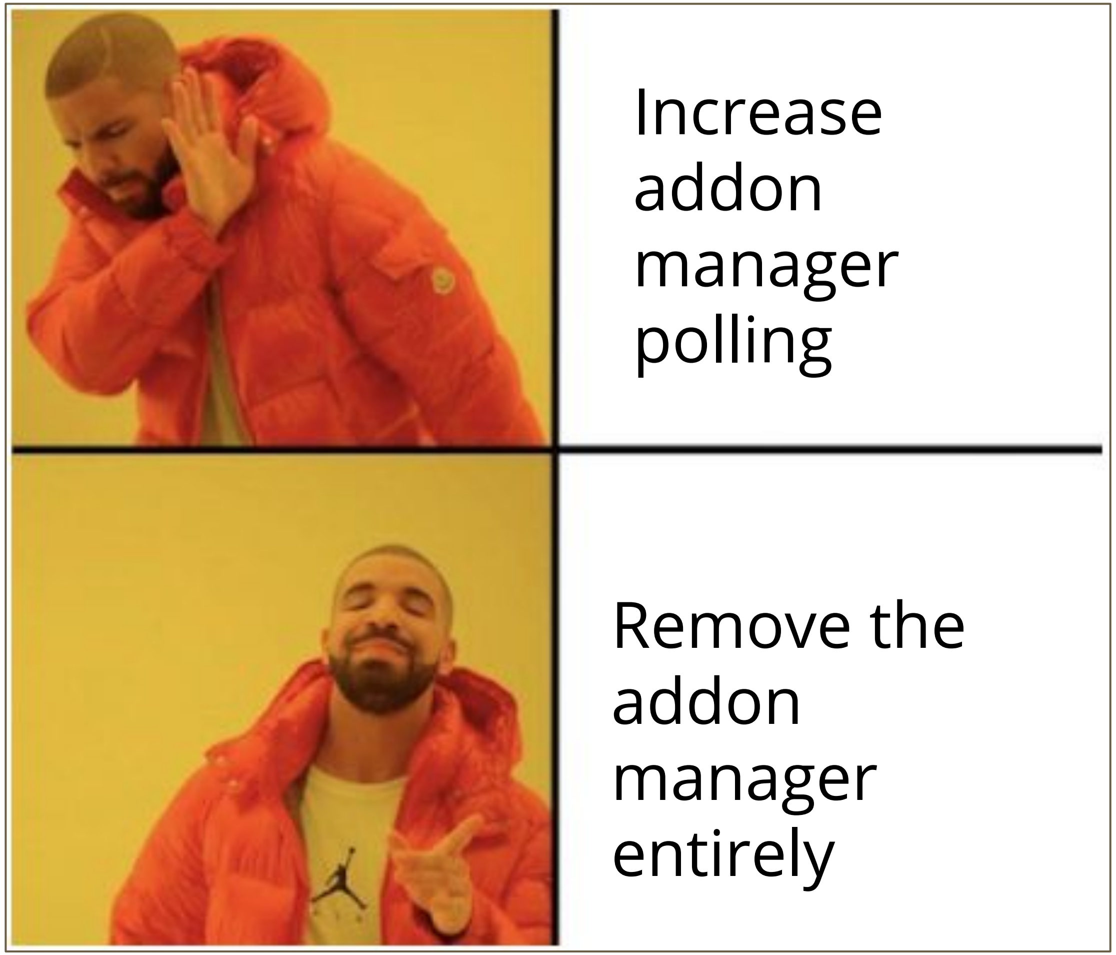
- Tried increasing poll time to see if that would improve overhead
  - It did!
- But, this created a tradeoff between poll time and user experience
- Ended up removing polling completely



Poll time: 30s



Polling removed


Increase addon manager polling
Remove the addon manager entirely

32%

reduction from removing addon manager

How is each part of k8s contributing to overhead?

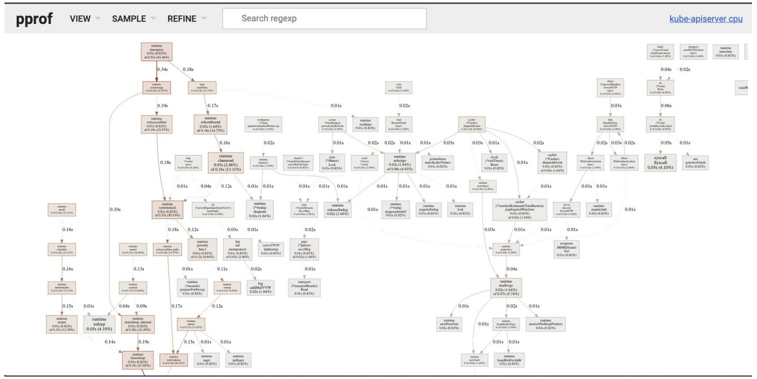
```
$ minikube ssh


$ pidstat 1 60
Average:      UID       PID    %usr %system  %guest   %wait    %CPU   CPU  Command
Average:        0      2103    0.53    0.22    0.00    0.00    0.75     -  dockerd
Average:        0      2110    0.08    0.03    0.00    0.00    0.12     -  containerd
Average:        0      3256    2.20    1.48    0.00    0.00    3.68     -  kube-apiserver
Average:        0      3272    0.85    0.63    0.00    0.20    1.48     -  kube-controller
Average:        0      3291    0.23    0.08    0.00    0.00    0.32     -  kube-scheduler
Average:        0      3318    0.80    0.67    0.00    0.00    1.47     -  etcd
Average:        0      3616    1.35    0.95    0.00    0.08    2.30     -  kubelet
Average:        0      4213    0.12    0.18    0.00    0.03    0.30     -  coredns
Average:        0      4325    0.10    0.12    0.00    0.00    0.22     -  coredns
Average:     1000      4455    0.05    0.28    0.00    0.03    0.33     -  sshd
Average:     1000      4496    0.20    0.43    0.00    0.37    0.63     -  pidstat
```

kube-apiserver overhead

pprof

- Go tool for visualizing and analyzing profiling data
- Tells you which functions are contributing to overhead and by how much

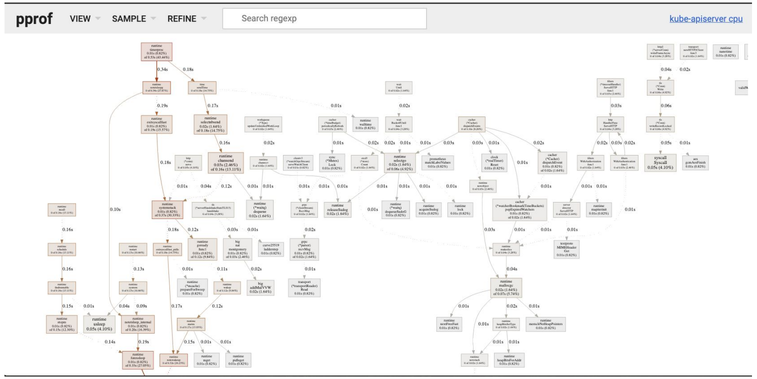
```
$ go tool pprof --http=":" localhost:[dashboard
port]/debug/pprof/profile?seconds=60

(pprof) top
Showing nodes accounting for 1010ms, 63.92% of 1580ms total
Showing top 10 nodes out of 403
      flat  flat%   sum%        cum   cum%
     620ms 39.24% 39.24%      620ms 39.24%  runtime.futex
     170ms 10.76% 50.00%      190ms 12.03%  syscall.Syscall
      60ms  3.80% 53.80%       60ms  3.80%  runtime.usleep
      40ms  2.53% 56.33%       40ms  2.53%
runtime.nextFreeFast
      20ms  1.27% 57.59%       20ms  1.27%
k8s.io/kubernetes/vendor/golang.org/x/net/http2.(*Framer).chec
kFrameOrder
```



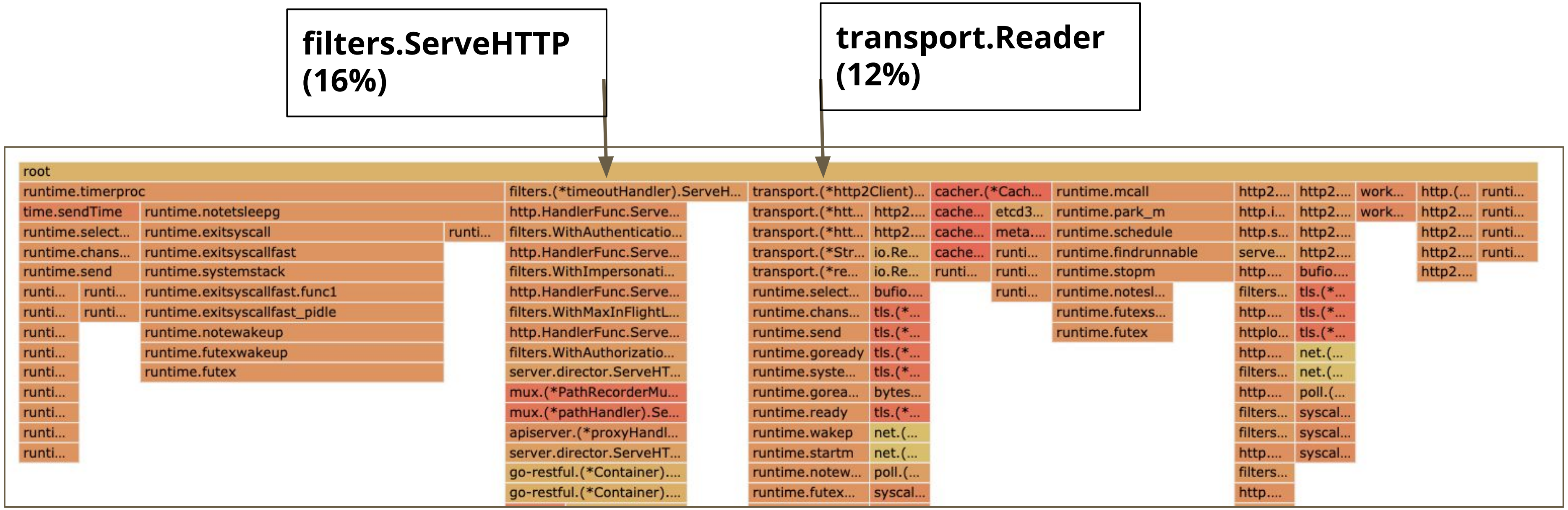
kube-apiserver pprof data

```
$ minikube start
$ minikube dashboard
```

```
$ go tool pprof --http=":" localhost:[dashboard
port]/debug/pprof/profile?seconds=60

(pprof) top
Showing nodes accounting for 1010ms, 63.92% of 1580ms total
Showing top 10 nodes out of 403
      flat  flat%   sum%        cum   cum%
     620ms 39.24% 39.24%      620ms 39.24%  runtime.futex
     170ms 10.76% 50.00%      190ms 12.03%  syscall.Syscall
      60ms  3.80% 53.80%       60ms  3.80%  runtime.usleep
      40ms  2.53% 56.33%       40ms  2.53%  runtime.nextFreeFast
      20ms  1.27% 57.59%       20ms  1.27%
k8s.io/kubernetes/vendor/golang.org/x/net/http2.(*Framer).checkFr
ameOrder
```



kube-apiserver flame graph



Leader election requests from scheduler & controller manager

```
$ minikube start --extra-config apiserver.v=10
$ kubectl logs kube-apiserver-minikube -n kube-system | grep -e GET | awk -F\" '{print $6}' | sort | uniq -c

11159
 396 Authorization: Bearer 5981c73a-be2b-4d18-9ca3-526baf4e9b13
 501 kube-apiserver/v1.18.3 (linux/amd64) kubernetes/2e7996e
 105 kube-controller-manager/v1.18.3 (linux/amd64) kubernetes/2e7996e/controller-discovery
 137 kube-controller-manager/v1.18.3 (linux/amd64) kubernetes/2e7996e/kube-controller-manager
 902 kube-controller-manager/v1.18.3 (linux/amd64) kubernetes/2e7996e/leader-election
 122 kube-controller-manager/v1.18.3 (linux/amd64)
kubernetes/2e7996e/system:serviceaccount:kube-system:cronjob-controller
 736 kube-controller-manager/v1.18.3 (linux/amd64)
kubernetes/2e7996e/system:serviceaccount:kube-system:generic-garbage-collector
 735 kube-controller-manager/v1.18.3 (linux/amd64)
kubernetes/2e7996e/system:serviceaccount:kube-system:resourcequota-controller
 901 kube-scheduler/v1.18.3 (linux/amd64) kubernetes/2e7996e/leader-election
 123 kubelet/v1.18.3 (linux/amd64) kubernetes/2e7996e
...
```

What’s leader election?

- Guarantees only one instance of kube-scheduler or kube-controller-manager is making decisions
- minikube by default is single node & only has one instance of each
- Can we turn leader election off?

--leader-elect=false

kube-controller-manager

--leader-elect Default: true
Start a leader election client and gain leadership before executing the main loop. Enable this when running replicated components for high availability.

kube-scheduler

--leader-elect Default: true
Start a leader election client and gain leadership before executing the main loop. Enable this when running replicated components for high availability.

18%

from turning off leader election requests & reducing coredns replicas to 1

NYPD
kube-controller-manager
--leader-elect=false
kube-scheduler
--leader-elect=false

etcd overhead

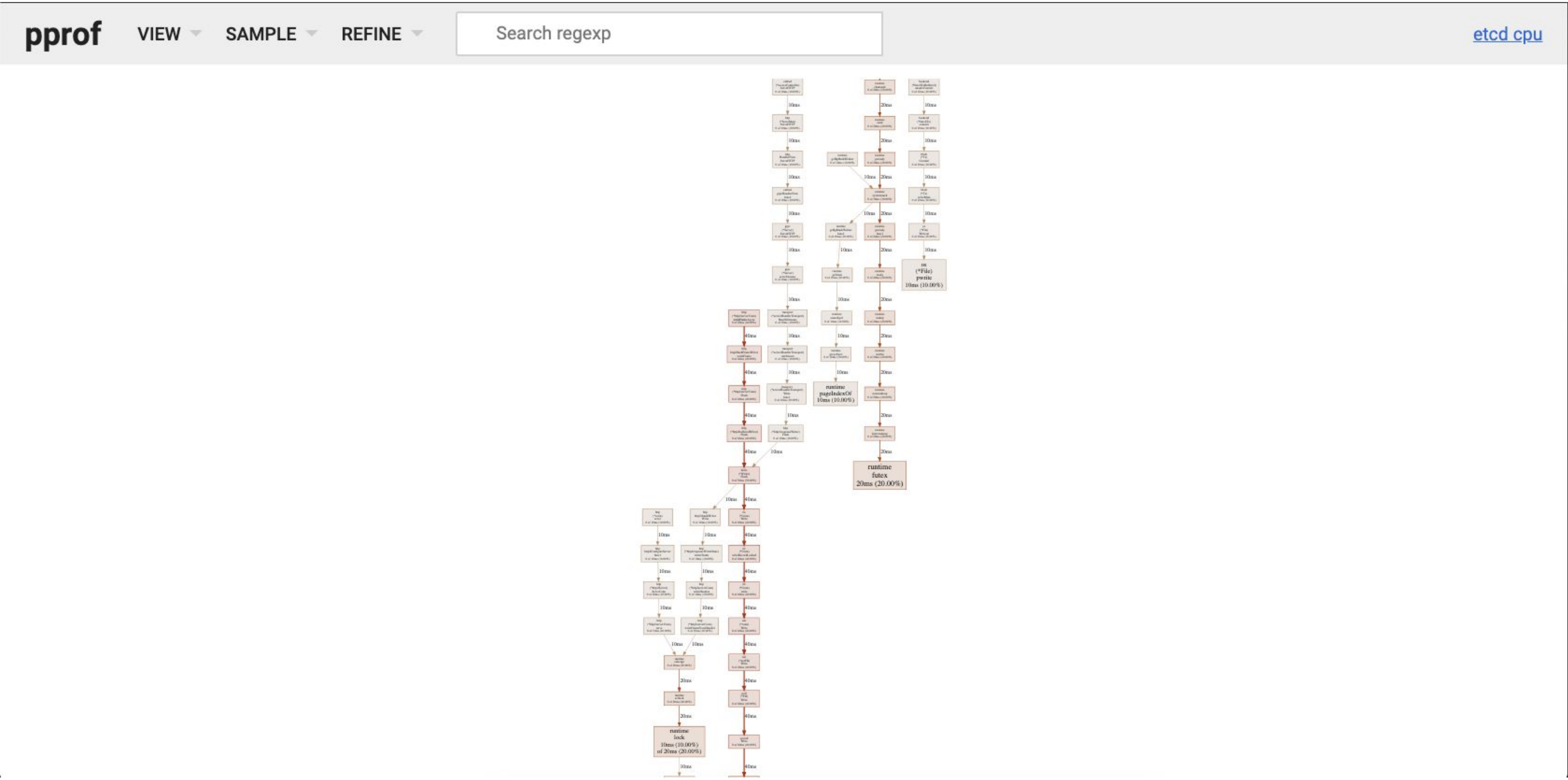
etcd logs

```
$ minikube start --extra-config etcd.debug=true
$ kubectl logs etcd-minikube -n kube-system

2020-07-16 15:58:37.165118 D | auth: found common name kube-apiserver-etcd-client
2020-07-16 15:58:37.165594 D | etcdserver/api/v3rpc: start time = 2020-07-16 15:58:37.165084652 +0000 UTC
m=+16.280593708, time spent = 495.61µs, remote = 127.0.0.1:49998, response type = /etcdserverpb.KV/Range, request count =
0, request size = 52, response count = 1, response size = 5930, request content =
key:"/registry/pods/kube-system/kube-apiserver-minikube"
2020-07-16 15:58:37.171748 D | auth: found common name kube-apiserver-etcd-client
...
```

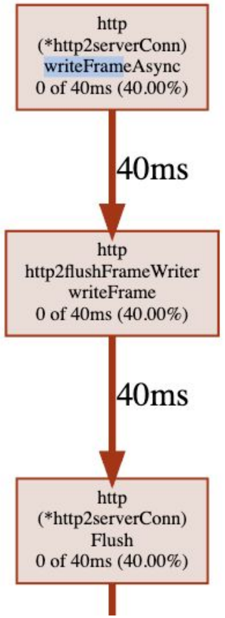
etcd pprof data

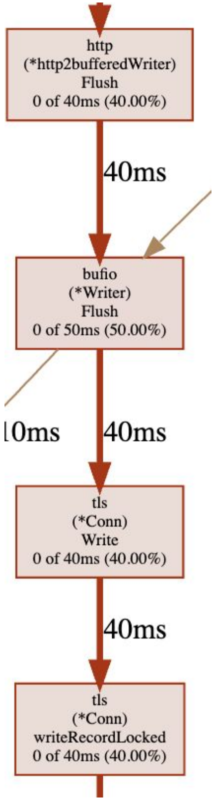
```
$ minikube start --extra-config etcd.enable-pprof=true --extra-config
etcd.listen-client-urls=https://127.0.0.1:2379,http://127.0.0.1:2382
$ kubectl port-forward po/etcd-minikube -n kube-system 8080:2382
```

```
$ go tool pprof --http=":" http://localhost:8080/debug/pprof/profile?seconds=60
File: etcd
Type: cpu
Time: Jun 10, 2020 at 8:54pm (EDT)
Duration: 1mins, Total samples = 840ms ( 1.40%)
Entering interactive mode (type "help" for commands, "o" for options)
(pprof) top
Showing nodes accounting for 560ms, 66.67% of 840ms total
Showing top 10 nodes out of 229
      flat  flat%   sum%        cum   cum%
      40ms 40.00% 40.00%       40ms 40.00%  syscall.Syscall
      20ms 20.00% 60.00%       20ms 20.00%  runtime.futex
      10ms 10.00% 70.00%       10ms 10.00%  os.(*File).pwrite
      10ms 10.00% 80.00%       10ms 10.00%  runtime.adjustpointers
      10ms 10.00% 90.00%       20ms 20.00%  runtime.lock
      10ms 10.00%   100%       10ms 10.00%  runtime.pageIndexOf
         0     0%   100%       50ms 50.00%  bufio.(*Writer).Flush
         0     0%   100%       40ms 40.00%  crypto/tls.(*Conn).Write
         0     0%   100%       40ms 40.00%  crypto/tls.(*Conn).write
         0     0%   100%       40ms 40.00%  crypto/tls.(*Conn).writeRecordLocked
```


pprof
VIEW
SAMPLE
REFINE
Search regexp
etcd cpu
os (*File) pwrite 10ms (10.00%)
runtime pageIndexOf 10ms (10.00%)
runtime futex 20ms (20.00%)
runtime lock 10ms (10.00%) of 20ms (20.00%)

writeFrameAsync





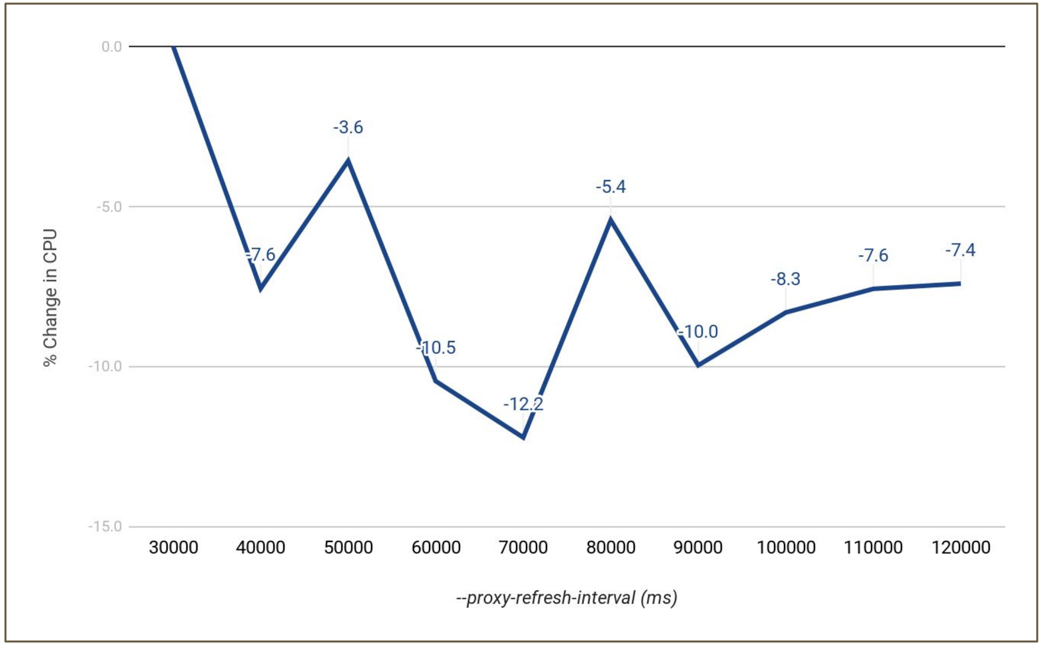
Searching through etcd code

- Looking for calls to Go's `http` library
- Found an `httpproxy` package in etcd code

```
// NewHandler creates a new HTTP handler, listening on the given transport,
// which will proxy requests to an etcd cluster.
// The handler will periodically update its view of the cluster.
func NewHandler(lg *zap.Logger, t *http.Transport, urlsFunc GetProxyURLs, failureWait time.Duration, refreshInterval time.Duration)
http.Handler {}
```

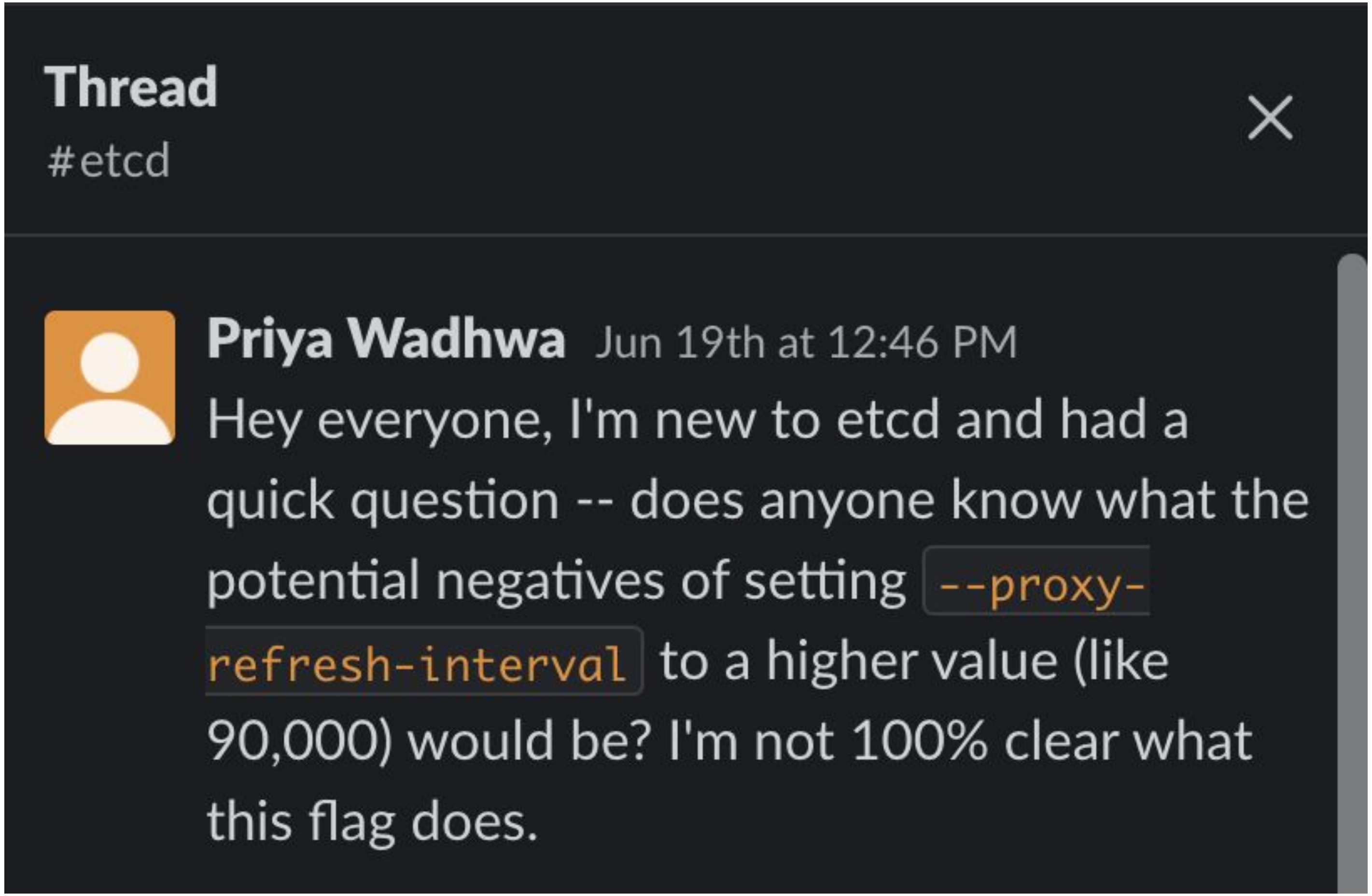
- `refreshInterval` is by default 30 seconds
- Set by the `--proxy-refresh-interval` flag

Hypothesis: Tuning --proxy-refresh-interval will improve overhead



What’s the tradeoff of increasing --proxy-refresh-interval?

- **–proxy-refresh-interval:** Time (in milliseconds) of the endpoints refresh interval (default: 30000)

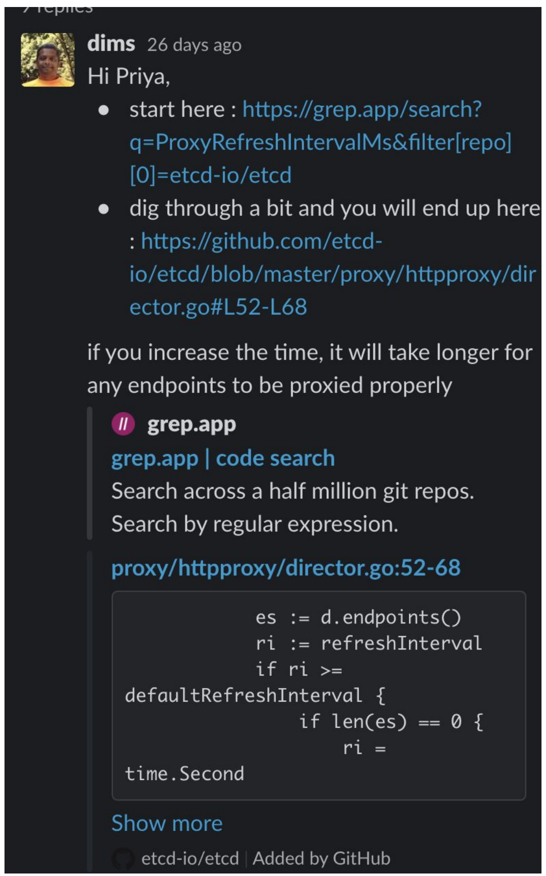


Thread

#etcd

Priya Wadhwa Jun 19th at 12:46 PM

Hey everyone, I'm new to etcd and had a quick question -- does anyone know what the potential negatives of setting `--proxy-refresh-interval` to a higher value (like 90,000) would be? I'm not 100% clear what this flag does.



dims 26 days ago

Hi Priya,

- start here : https://grep.app/search?q=ProxyRefreshIntervalMs&filter[repo][0]=etcd-io/etcd
- dig through a bit and you will end up here : https://github.com/etcd-io/etcd/blob/master/proxy/httpproxy/director.go#L52-L68

if you increase the time, it will take longer for any endpoints to be proxied properly

grep.app

grep.app | code search

Search across a half million git repos. Search by regular expression.

proxy/httpproxy/director.go:52-68

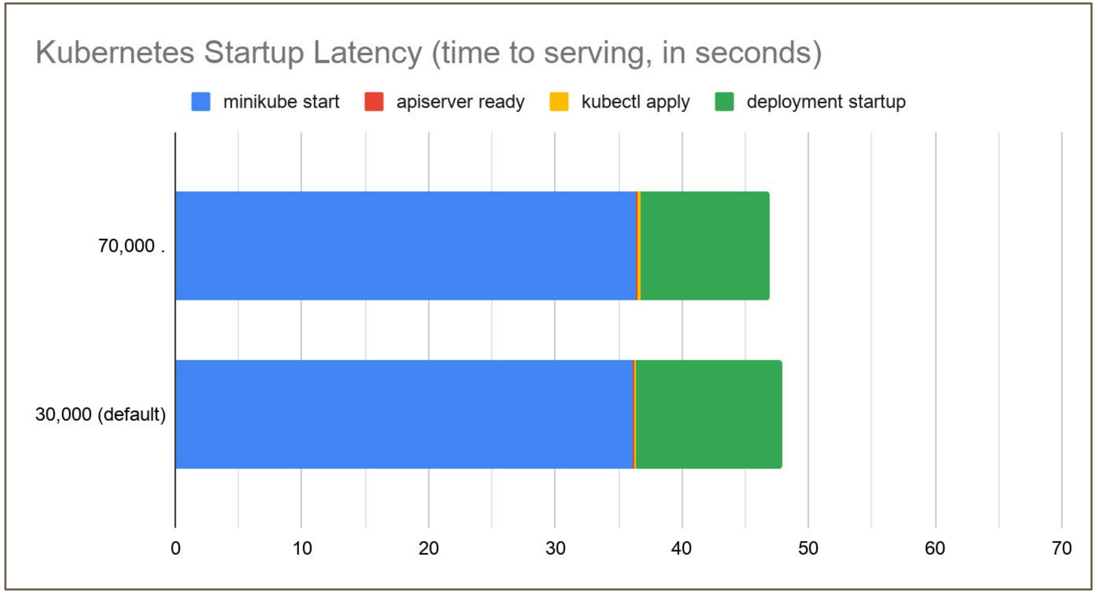
```
            es := d.endpoints()
            ri := refreshInterval
            if ri >=
defaultRefreshInterval {
                if len(es) == 0 {
                    ri =
time.Second
```

Show more

etcd-io/etcd | Added by GitHub

Does changing the refresh interval make user experience worse?

- Tested UX by measuring start time -> successful application deployment

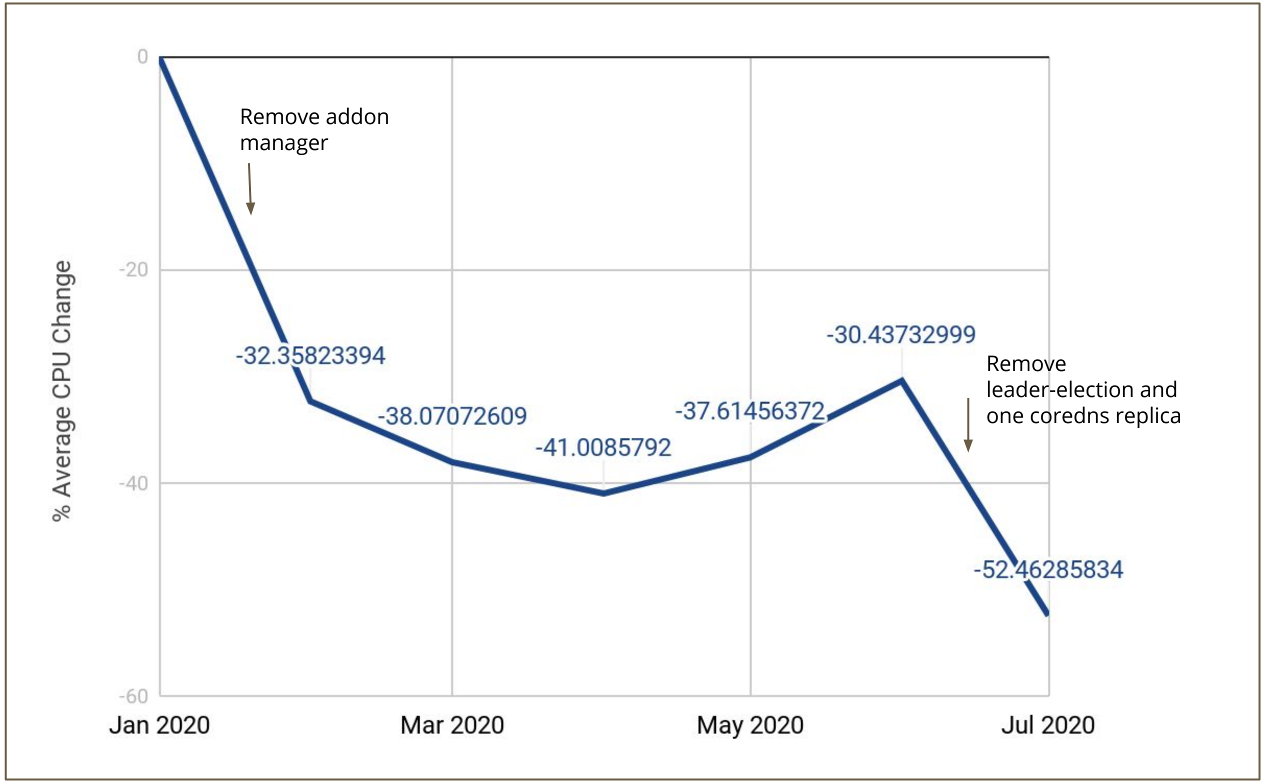


4%

Reduction from increasing --proxy-refresh-interval=70000

🙂 🔥 🎉

Minikube overhead in 2020



Takeaways

- Removing unnecessary work is great!
  - Addon manager
  - Coredns pod
  - Leader election
- Consider the tradeoff between overhead and user experience
- Collaboration is really important

Thank you!

- Brendan Gregg - www.brendangregg.com
  - Super helpful in learning the basics of improving performance
  - How to read flame graphs, use BCC tools, Linux perf_events
- Dave Cheney - dave.cheney.net
  - https://dave.cheney.net/high-performance-go-workshop/dotgo-paris.html#profiling
  - Learned a lot about improving performance in Go applications
  - How to collect and read pprof data

Related Talks

- Minikube Deep Dive - **Wednesday**, August 19 • 13:00 - 13:35
- Performance Optimization - Rook on Kubernetes - **Thursday**, August 20 • 14:30 - 15:05