CernVM-FS Profiling

Razvan-Nicolae Virtan

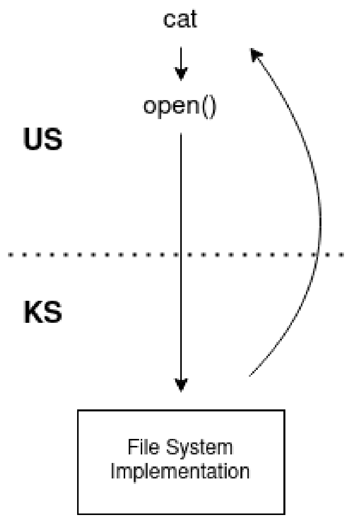
Supervisors: Radu Popescu, Jakob Blomer

About CernVM-FS *(CVMFS)*

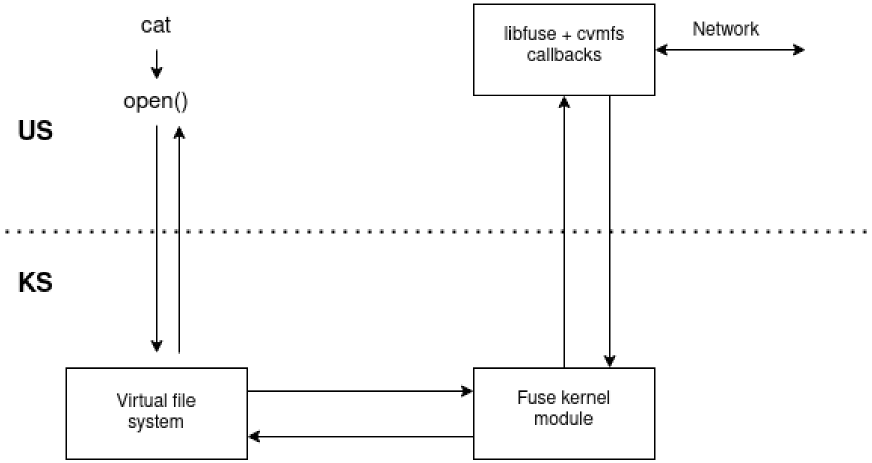
- created and optimized to deliver scientific software stacks to a distributed compute infrastructure
- offers a file system interface for software repositories

```
[razvan@~]$ cat /cvmfs/atlas.cern.ch/repo/test
```
# **new cvmfs process created on local system**

Classic scenario
(i.e. ext4)



CVMFS



Why profiling?

Benchmark scenario:

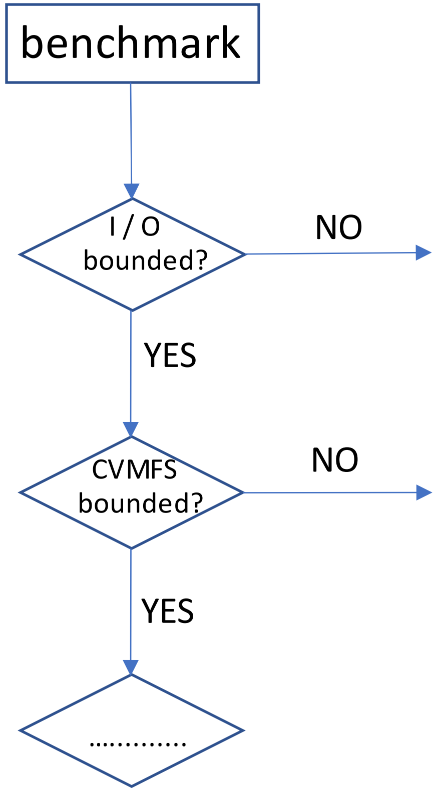
- Multi core system
- Cold Cache
- Multiple processes, attempting to access different data from the same repo (e. g. different jobs using different software versions)

Performance on a synthetic benchmark:

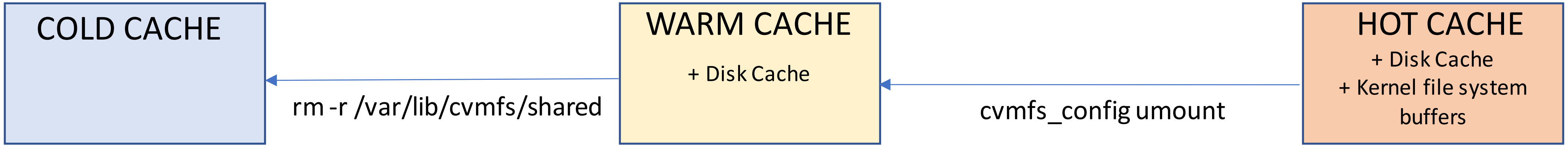
CVMFS	Local filesystem
1.38 GB for 1 min	3.97 GB for 1 min

CVMFS Profiling Goals

1. Develop a general set of tools & procedures for analyzing CVMFS performance
2. Apply these tools on some known benchmarks and spot possible bottlenecks



I / O bounded? New tool: avg_cache_time.sh



COLD CACHE

rm -r /var/lib/cvmfs/shared

WARM CACHE

+ Disk Cache

cvmfs_config umount

HOT CACHE

+ Disk Cache
+ Kernel file system buffers

```
$ ./profiling_tools/avg_cache_time.sh --rounds 2 ./tensorflow_benchmark.sh

Cache_Type        Real Avg (s)    User Avg(s)     Sys Avg(s)
cold                25.560          10.639        1.282
warm                12.924          10.359        1.183
hot                 11.690          10.051        0.959
-------------------------------------------------------
Cache_Type        CPU             BLOCKED
cold              0.399             0.601
warm              0.893             0.107
hot               0.942             0.058
-------------------------------------------------------
Compare Case       Real Time      User Time      Sys Time
cold / hot           2.192          1.058          1.337
cold / warm          1.992          1.027          1.082
warm / hot           1.106          1.031          1.233
```

CVMFS bounded? cvmfs_talk

- How much of the blocked time is actually spent in CVMFS?
- Newly added counter to the set of cvmfs internal profiling counters
- Measure time spent in cvmfs callbacks and calculate the total

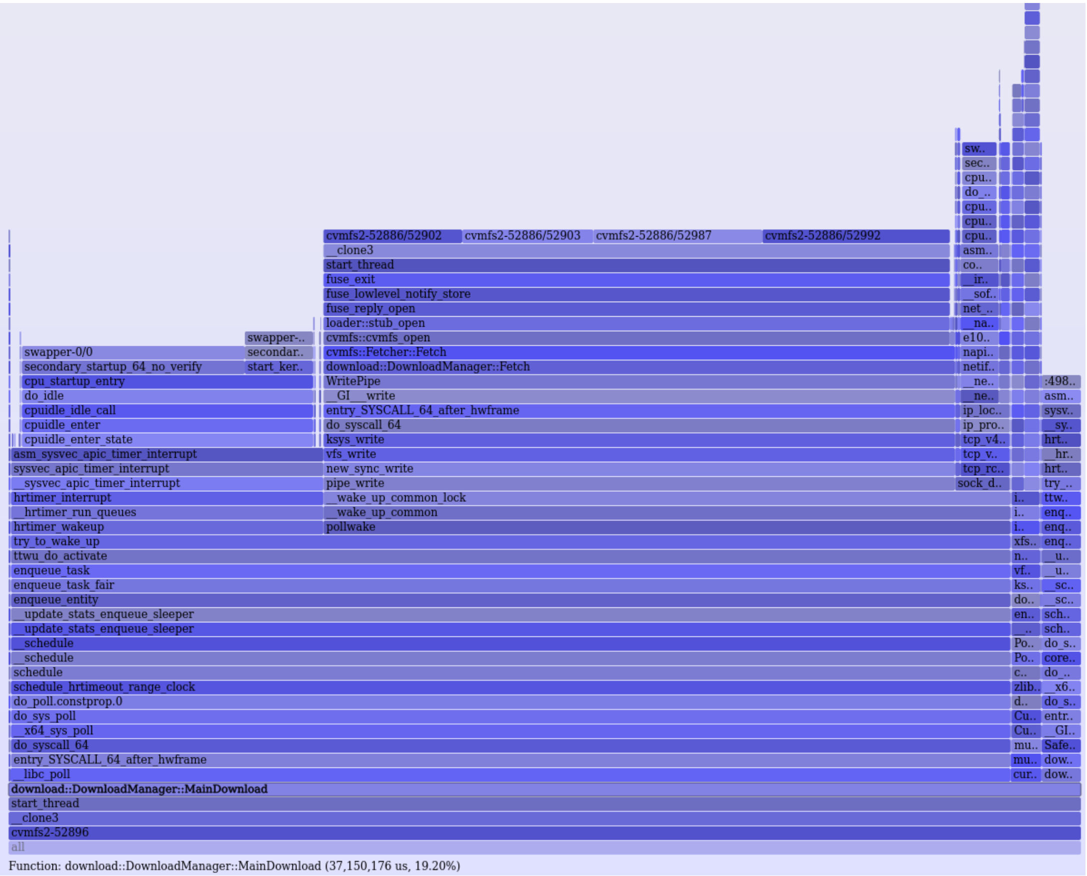
$$T_{\text{Callbacks}} \;/\; T_{\text{Blocked}}$$

```
$ cvmfs_talk -i unpacked.cern.ch internal affairs

….

Total Time In Callbacks = 34945ms

….
```

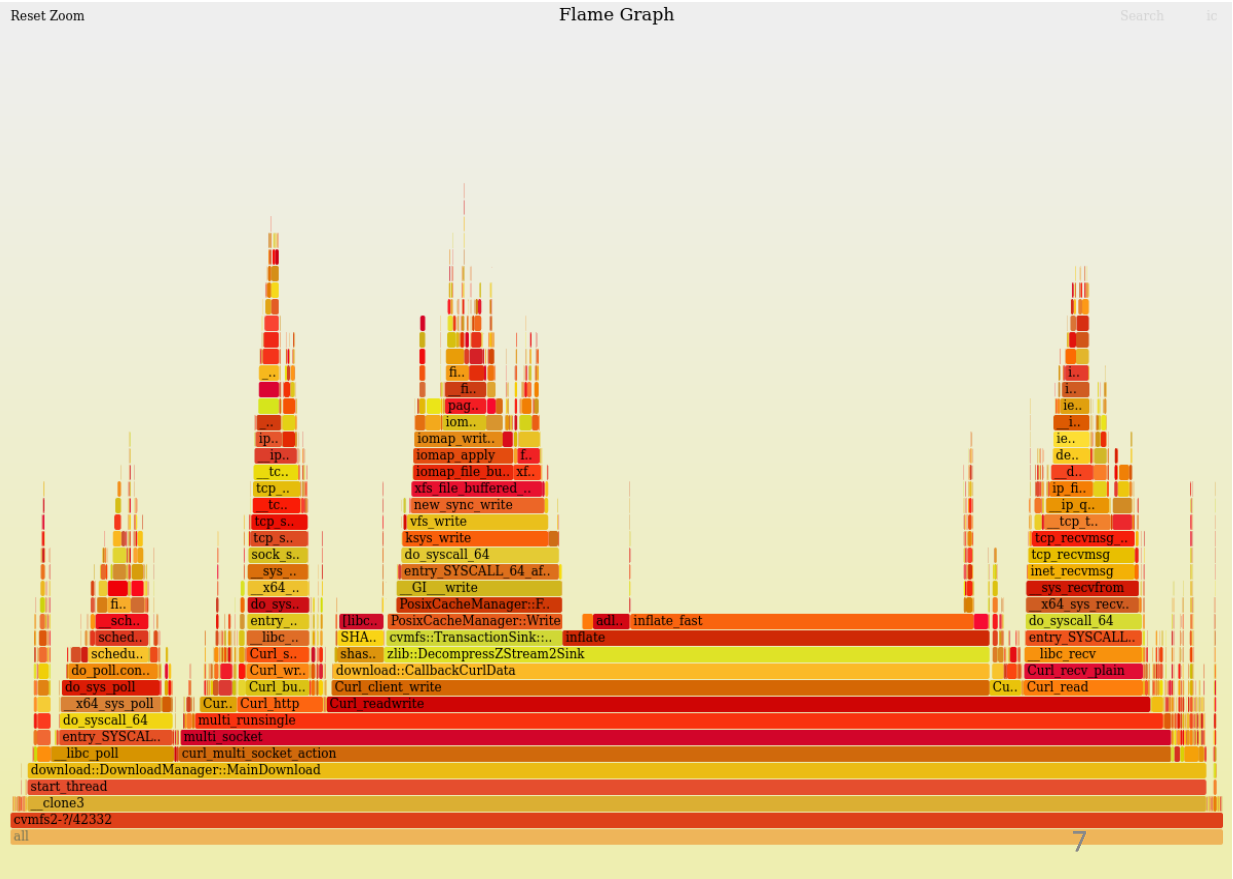
Where is the bottleneck? New tool: generate_flamegraphs.sh

- **Method 1**: exhaustive ON / OFF CPU analysis using flamegraphs

```
$ ./profiling_tools/generate_flamegraphs.sh --oncpu --dwarf --benchmark lhcb_benchmark.sh --cache hot
```



Function: download::DownloadManager::MainDownload (37,150,176 us, 19.20%)



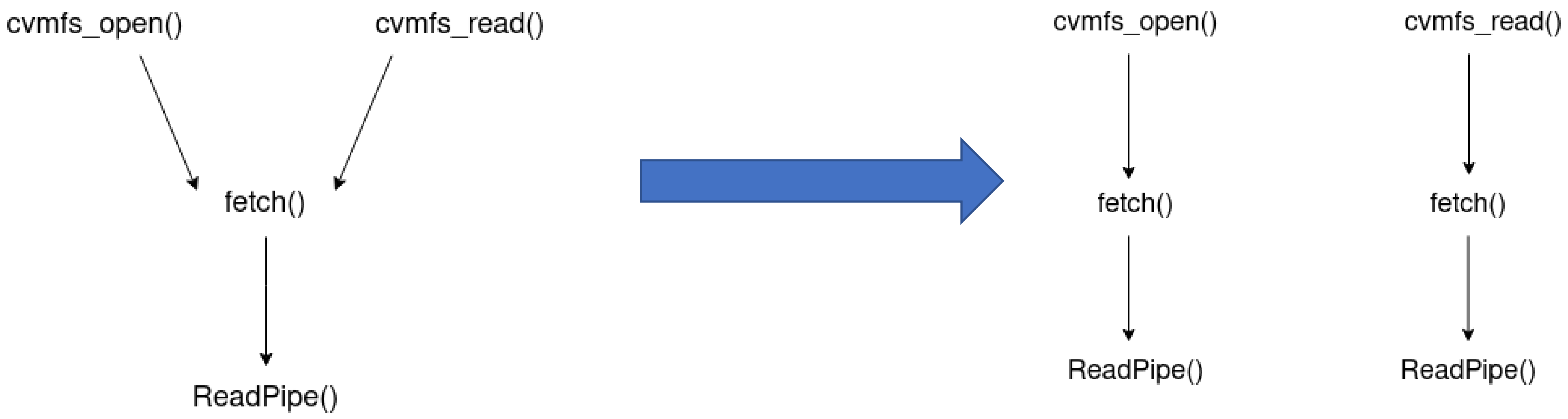
Where is the bottleneck? cvmfs_talk

- Flamegraphs are inaccurate / expensive and can fail on multi-threaded scenarios.

- **Method 2**:
  - add more timers in the CVMFS code, in places that can become bottlenecks
  - generate partial flamegraphs internally, for relevant parts of the CVMFS code

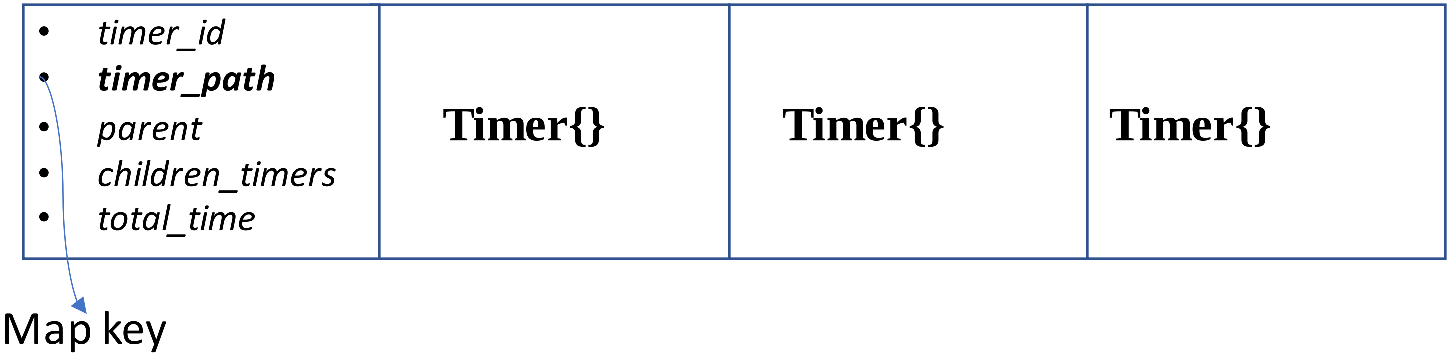
=> New timers interface in CVMFS

Timer Requirements



- The same timer, reached from two different call stacks, needs different records.

Timer implementation



- *timer_id*
- ***timer_path***
- *parent*
- *children_timers*
- *total_time*

Timer{}

Timer{}

Timer{}

Map key

Timers Map + *last_started_timer*

timer_path = (last_started_timer->timer_path << 8) + timer_id; → MAX 256 timers and 8 timers in a call trace

TimerTree{} class – singleton, only initialization + reporting

RAII timer interface

- **TimerGuard** is a wrapper object for the **Timer** backend

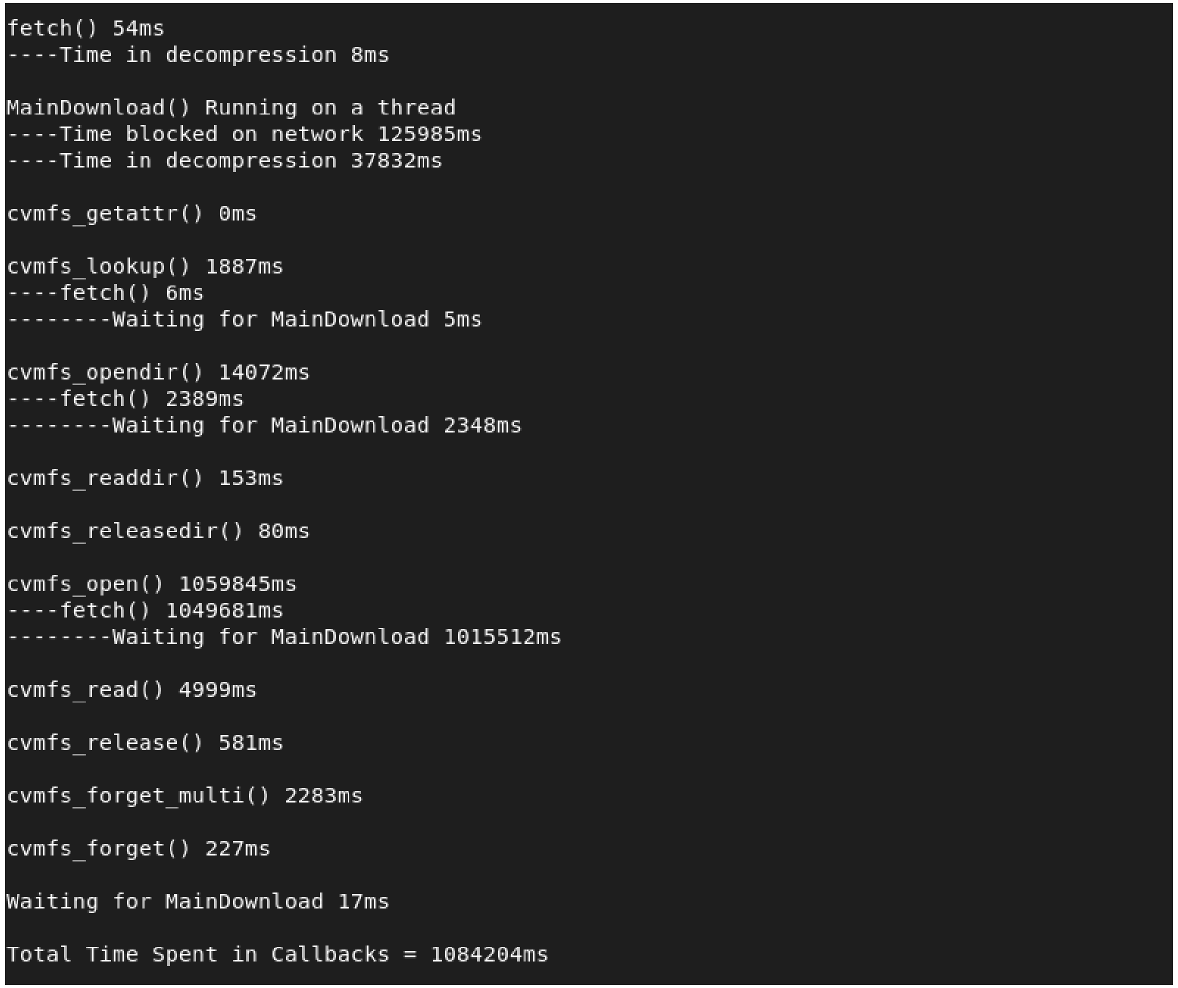
TimerGuard()	**~TimerGuard()**
• search / add timer to map • record t_0	• compute *dt* • add *dt* to the total time

- ***Some usage examples:***

```
cvmfs_open() {
  TimerGuard timer_guard("cvmfs_open()",    CVMFS_OPEN_TIMER, ...);
  ......
}
```

```
......
{
   TimerGuard timer_guard(...);
   retval = DecompressZStream2Sink(...);
   if ( retval == zlib::kStreamDataError ) {
      ......
   }
}
.....
```

Output example



```
fetch() 54ms
----Time in decompression 8ms

MainDownload() Running on a thread
----Time blocked on network 125985ms
----Time in decompression 37832ms

cvmfs_getattr() 0ms

cvmfs_lookup() 1887ms
----fetch() 6ms
--------Waiting for MainDownload 5ms

cvmfs_opendir() 14072ms
----fetch() 2389ms
--------Waiting for MainDownload 2348ms

cvmfs_readdir() 153ms

cvmfs_releasedir() 80ms

cvmfs_open() 1059845ms
----fetch() 1049681ms
--------Waiting for MainDownload 1015512ms

cvmfs_read() 4999ms

cvmfs_release() 581ms

cvmfs_forget_multi() 2283ms

cvmfs_forget() 227ms

Waiting for MainDownload 17ms

Total Time Spent in Callbacks = 1084204ms
```

Synchronization issues

- 2 threads attempting to create the same timer
  - One lock associated for the whole map

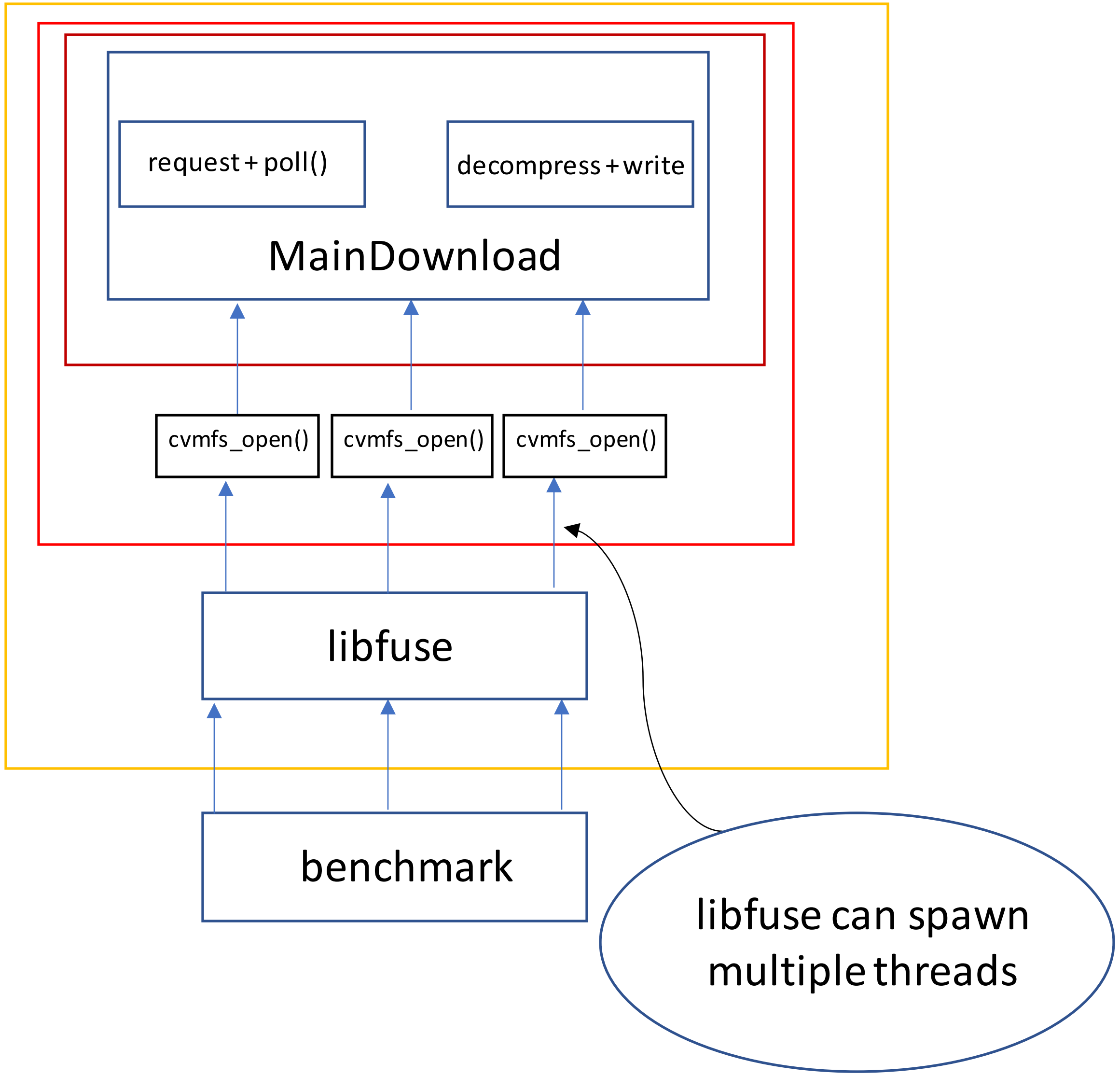
- 2 threads attempting to modify the same timer

  - *timer_id*
  - *timer_path*
  - *parent*
  - *children_timers*
  - *total_time* ***atomic***

- Each thread needs it's own *last_started_timer*
  - Put *last_started_timer* in TLS

Benchmark scenario:

- Multi core system
- Cold Cache
- Multiple processes, attempting to access different data from the same repo (e. g. different jobs using different software versions)



request + poll()

decompress + write

MainDownload

cvmfs_open()

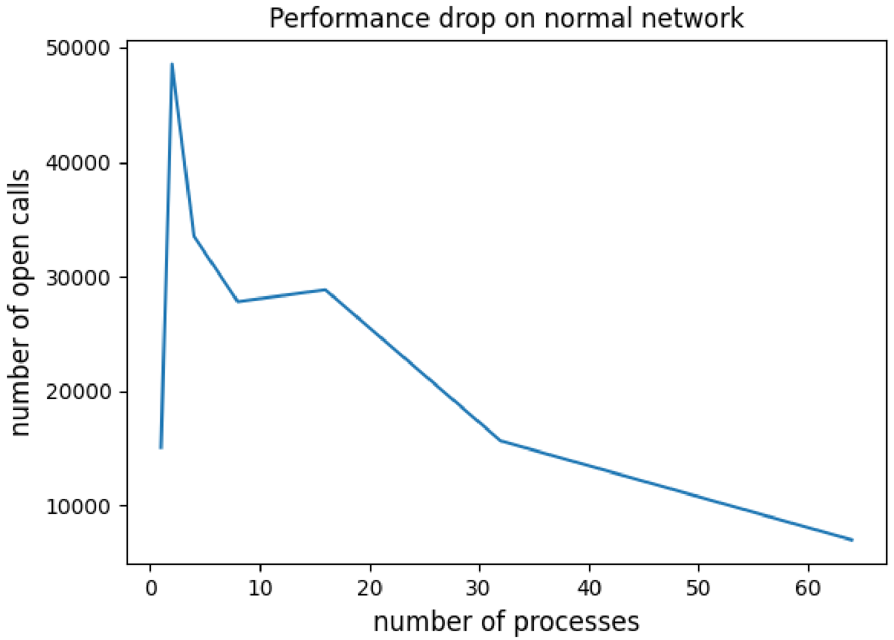
cvmfs_open()

cvmfs_open()

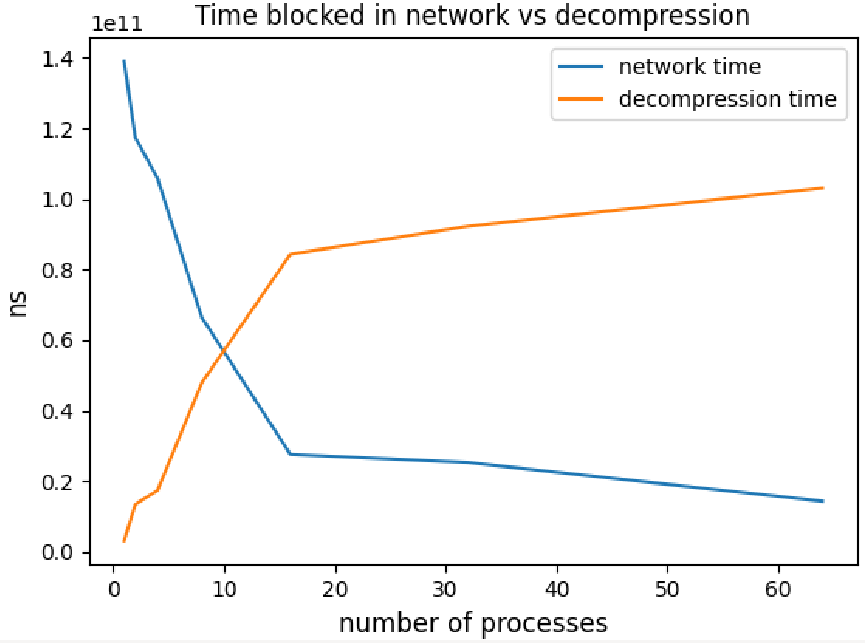
libfuse

benchmark

libfuse can spawn multiple threads

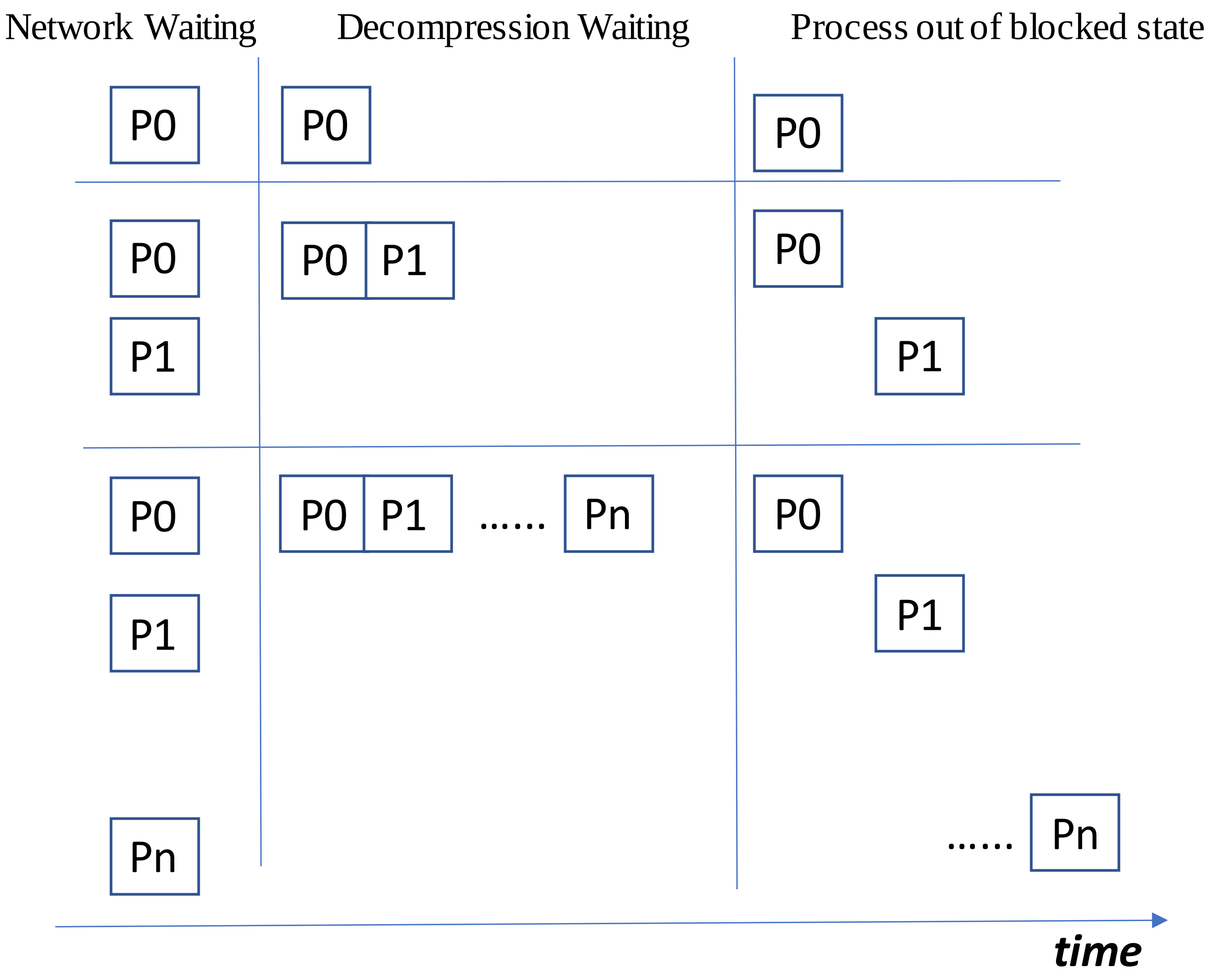


Performance drop on normal network



Time blocked in network vs decompression

The reason behind the bottleneck



Network Waiting

Decompression Waiting

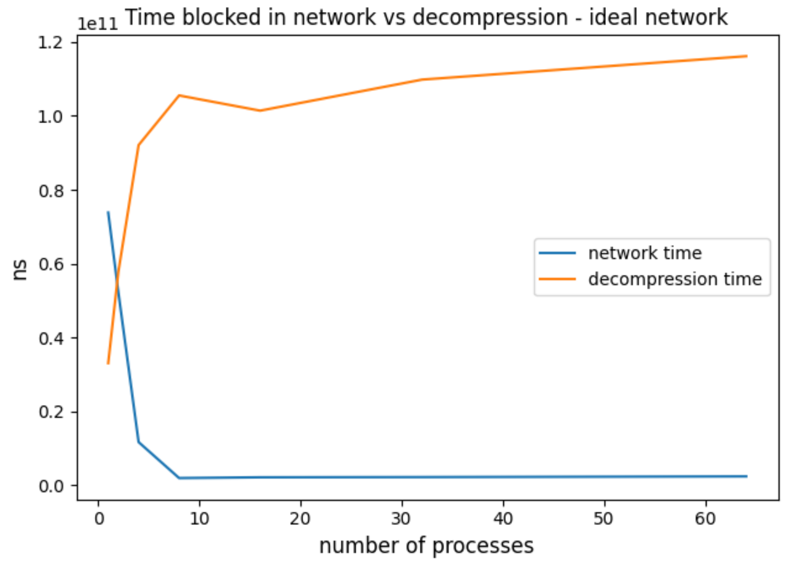
Process out of blocked state

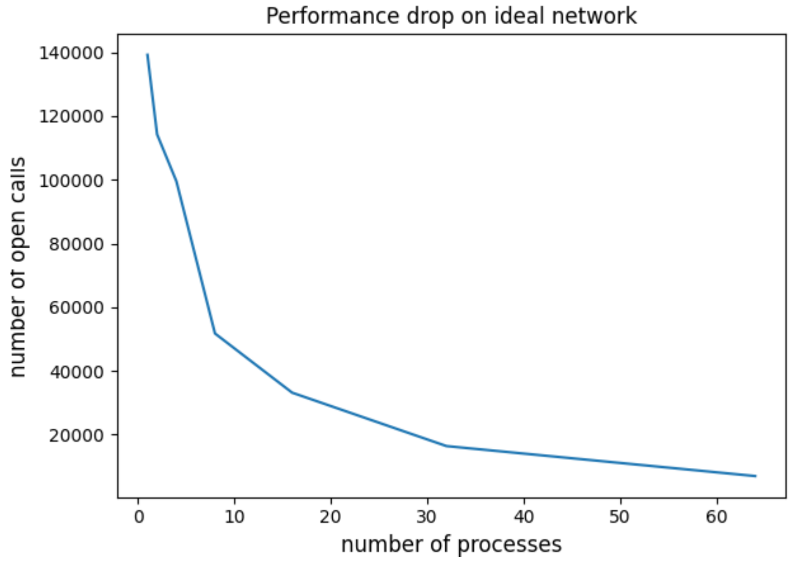
P0 | P0 | P0

P0, P1 | P0 P1 | P0, P1

P0, P1, Pn | P0 P1 …… Pn | P0, P1, …… Pn

time





Processes queuing for decompression

Conclusions

- Profiling tools are essential for further improvements in CVMFS performance.
- We developed external tools for the first steps of the analysis.
- We developed internal timers that can offer an in-depth view on the possible bottlenecks.
- We found that, by parallelizing the data decompression, we can improve performance on multiple-processes / multiple-data scenarios.
- We are currently looking into a new benchmark, provided by the Alice experiment.