LF *live*

MENTORSHIP SERIES

LF live MENTORSHIP SERIES

Memoptimizer watches your memory usage so you don’t have to!

Khalid Aziz, Senior Software Engineer, Oracle

LF live MENTORSHIP SERIES

Agenda

- The problem we are solving
- Reduce sys admin burden through automation
- Does it work?

Kernel runs out of free pages

- System has large amount of memory, does not use memory heavily, just does lot of I/O
- You start to see messages from kernel it has run out of order n pages, or launching a new program takes a long time
- You check memory usage:

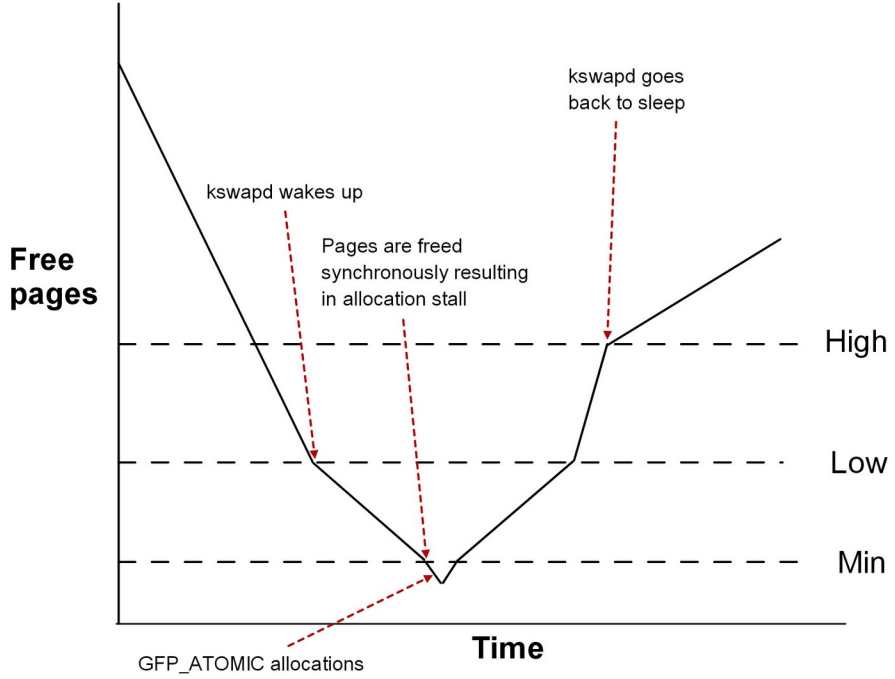
```
# free -h
              total        used        free      shared  buff/cache   available
Mem:           722G        2.7G        3.5G         33M        716G        715G
Swap:          4.0G        524K        4.0G

# cat /proc/buddyinfo
Node 0, zone      DMA      0      1      1      2      3      0      1      0      1      1      3
Node 0, zone    DMA32      8      4      8      5      5      7      9      7      5      6    289
Node 0, zone   Normal    291    259    768    533    284   1923     34      9    583      1      0
Node 1, zone   Normal   1239   1647    577    428    627    162    231    158    360      0      0
```

- On a system with 722G memory, only 2.7G is in use and yet only 3.5G of memory is free

Where did all memory go?

- Linux kernel uses page/buffer cache to cache frequently used data, primarily disk resident data
- Kernel allocates as much of unused memory as possible to page/buffer cache
- In previous scenario 716G of memory is allocated to page/buffer cache and the number of pages of order 3 and higher is fairly low
- Memory used by page/buffer cache can be reclaimed by kernel whenever more memory is needed which is done asynchronously in the background by kswapd
- Pages reclaimed from page/buffer cache can be compacted into higher order pages. This done by kcompactd
- What triggers reclamation/compaction is the watermarks

What are watermarks



Based upon picture in Mel Gorman's book -
https://www.kernel.org/doc/gorman/html/understand/understand005.html

Setting watermarks

- Min watermark is computed so kernel will keep a minimum number of pages free which are reserved for critical allocations (GFP_ATOMIC). All other allocations wait until synchronous (direct) reclamation can free up pages
- Min watermark can be changed through tunable `min_free_kbytes` (mfk)
- Low and high watermarks are computed by adding an offset to min watermark calculated in `__setup_per_zone_wmarks()`
- Offset added to min watermark to compute low and high is the larger of 4 times of min, and `watermark_scale_factor` fraction of number of pages
- The gap between watermarks can be changed through `watermark_scale_factor`
- Just the low and high watermarks can be changed by changing `watermark_scale_factor` or all three can be changed by changing `min_free_kbytes`

Playing with watermarks

- Min watermark had been capped at 64M back in 2004-2005 timeframe (pre-git days) irrespective of size of system memory. Finally raised to 256M in 2020
- A customer runs into scenario described earlier and the obvious solution is to get page reclamation to kick in earlier. So we raise mfk as mitigation measure
- This works for a while until a new workload size/behavior breaks this. So we raise mfk yet again. We got to the point of following recommendations for mfk:

Memory size (GB)	16	192	384	768	1024	1536	2048
mfk	82M	960M	1.92G	3.84G	5.12G	7.68G	10.24G

Time to rethink

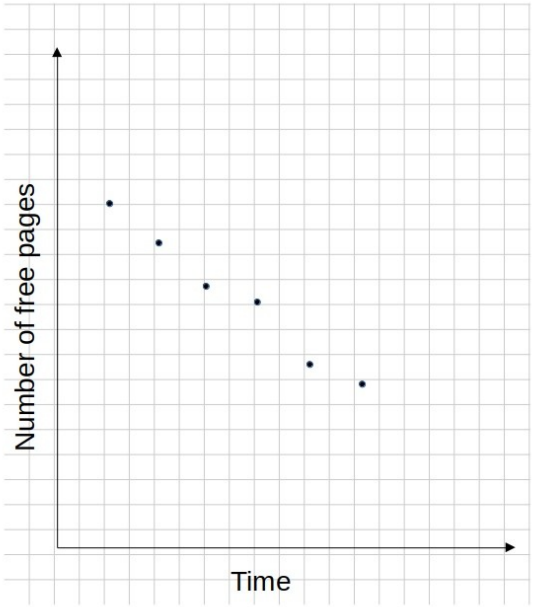
- Raising mfk continually is not a sustainable solution
- We need a more rational cap on mfk in kernel. mfk cap was raised after a very long time to 256M in kernel in 2020
- Watermarks need to be tuned to workload behavior on each system. Workload behavior can change from system to system and from time to time on the same system
- Forcing more aggressive reclamation during periods of high page allocation/free activity by raising watermarks works but when the system calms down, keeping watermarks high results in fewer pages available to page/buffer cache

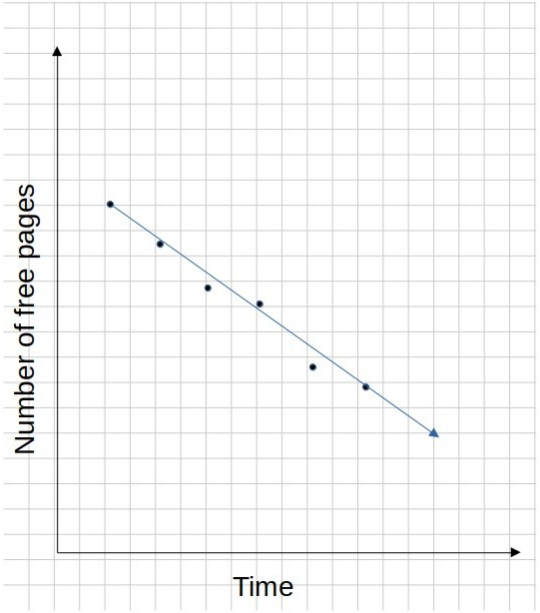
Adaptive and Proactive watermarks

- Kernel behaves reactively to memory pressure
- By the time kernel sees low memory condition, it may already be under too severe a stress to recover. It may be too late to avert impending allocation stalls
- What if we could study system behavior continuously, model it mathematically, project future free memory needs based upon that model and tune watermarks to address upcoming memory needs?
- We can model system behavior using recent free page data and calculate a trend line for memory consumption using the method of least squares
- This mathematical formula x = ay + b (where x is number of pages, y is time) then allows us to calculate memory demand at any point in future given current memory consumption trend
- We can also calculate current page reclamation and compaction rates by watching change in number of free pages of various order over time
- With this information in hand, we can calculate if the system will run out of free pages or higher order pages in near future. If so, we can adjust watermarks to mitigate this

LF live MENTORSHIP SERIES

Method of least squares





Putting it all together - Modeling

- Current number of free pages give us the points on the plot which we can fit a trend line to
- For the trend line to track system behavior dynamically, it must be recomputed periodically
- Create a sliding window of last N data points
- Sample system periodically, fill the sliding window and fit a trend line to these points using method of least squares
- Create forecasts for memory exhaustion using this trend line for each zone and overall system

Putting it all together - Actions

- If there is impending free page exhaustion, force earlier and/or longer reclamation by adjusting watermarks
- If higher order pages are projected to run out because of memory fragmentation, force compaction
- If system activity is slowing down (free page count going up), back off aggressive reclamation by lowering watermarks
- At the next sampling period, update sliding window and repeat the process

Prediction in kernel

- Initial implementation was chosen to be a kernel patch to kswapd.
- Kswapd could run the modeling algorithm every time it runs
- If the result of prediction based on system modeling indicates potential free pages exhaustion in near future, it could use reclaim boost to reclaim more pages
- If the result of prediction indicates potential to run out of higher order pages in near future, kswapd would wake kcompactd after finishing its reclamation run
- This was launched as LF Linux Kernel Mentorship project in summer 2019
- Working with mentees, we sent patches to LKML
- Based upon community feedback, we decided to implement this as an external daemon instead

memoptimizer* daemon

- This new implementation is realized in memoptimizer daemon
- memoptimizer uses a sliding window of size 8
- This has been launched as an open source github project at https://github.com/oracle/memoptimizer (contributions welcomed)
- memoptimizer behavior can be modified using command line options or the configuration file `/etc/sysconfig/memoptimizer` or `/etc/default/memoptimizer`
- It uses syslog logging facility. Verbosity level for logging can be changed with “-v” option or with “VERBOSE” parameter in configuration file
- With verbosity level of 5, memoptimizer will log all its actions along with reasons for those actions and reclamation and compaction rates it computes periodically

* *Name might change in future*

Data sources and control knobs

- Implementing trend analysis requires data sources and system tuning dynamically requires available knobs
- Linux kernel provides following data sources:
  `/proc/vmstat` : page reclamation and cache page usage
  `/proc/buddyinfo` : per zone count of free pages of various orders
  `/proc/zoneinfo` : per zone watermarks
- Kernel also provides following knobs to tune free memory management:
  `/proc/sys/vm/watermark_scale_factor`: Scale factor for gap between watermarks
  `/sys/devices/system/node/node%d/compact`: Force compaction on a NUMA node

memoptimizer data and actions

- memoptimizer gets free page count for each node for each order from `/proc/buddyinfo` and uses this data to compute a trend line for each order page for each zone
- Current per zone watermarks are read from `/proc/zoneinfo`
- memoptimizer computes current reclamation and compaction rates and uses those values in predicting potential exhaustion of order 0 and higher pages.
- Reclamation rate is computed from reclaimed pages reported in `/proc/vmstat`. Compaction rate is computed by looking at the change in number of higher order pages in `/proc/buddyinfo`
- memoptimizer forces longer reclamation by scaling up watermarks by writing a new value to `/proc/sys/vm/watermark_scale_factor`
- It forces compaction on a node by writing to `/sys/devices/system/node/node%d/compact`

Test and Metric

- A workload that performs significant amount of I/O, especially file I/O where files are created and destroyed frequently can cause significant number of reclaimable pages tied up in page/buffer cache
- A workload comprised of 9 parallel dd to SSDs and a kernel compile with “make -j60” on a 96 processor system with 768GB of memory creates a suitable workload for testing effectiveness of memoptimizer daemon
- memoptimizer attempts to minimize the number of allocation/compaction failures as well as stalls. Number of allocation and compaction stalls over a test run gives a good insight into its effectiveness

Proof is in the pudding

- With the previously identified workload, tests were run without memoptimizer and then with memoptimizer running. Number of stalls as reported by `/proc/vmstat` were measured and recorded over a roughly 140 minute period
- Now for the result:

	No memoptimizer	Memoptimizer running
4.1.12 (UEK4)	5529	623
4.14.35 (UEK5)	3216	42
5.4.17 (UEK6)	212	1
kernel.org 5.14	190	0

Doing more with memoptimizer

- memoptimizer monitors system changes and models system behavior. This can have further uses
- Memoptimizer has evolved to include capabilities to:
  - Make a one time change to a system tunable at startup
  - Update values of system tunables upon system events, e.g. change in number of hugepages

Future possibilities for memoptimizer

- Make use of `/proc/sys/vm/compaction_proactiveness` to force gentler compaction
- Add more knobs under `/proc/sys/vm` for automated tuning (for example `watermark_boost_factor`, `swappiness`, `min_free_kbytes`, others…)
- Interaction with cgroups
- Possibly use data gathered by DAMON and PSI

LF live MENTORSHIP SERIES

Thank you for joining us today!

We hope it will be helpful in your journey to learning more about effective and productive participation in open source projects. We will leave you with a few additional resources for your continued learning:

- The LF Mentoring Program is designed to help new developers with necessary skills and resources to experiment, learn and contribute effectively to open source communities.
- Outreachy remote internships program supports diversity in open source and free software
- Linux Foundation Training offers a wide range of free courses, webinars, tutorials and publications to help you explore the open source technology landscape.
- Linux Foundation Events also provide educational content across a range of skill levels and topics, as well as the chance to meet others in the community, to collaborate, exchange ideas, expand job opportunities and more. You can find all events at events.linuxfoundation.org.