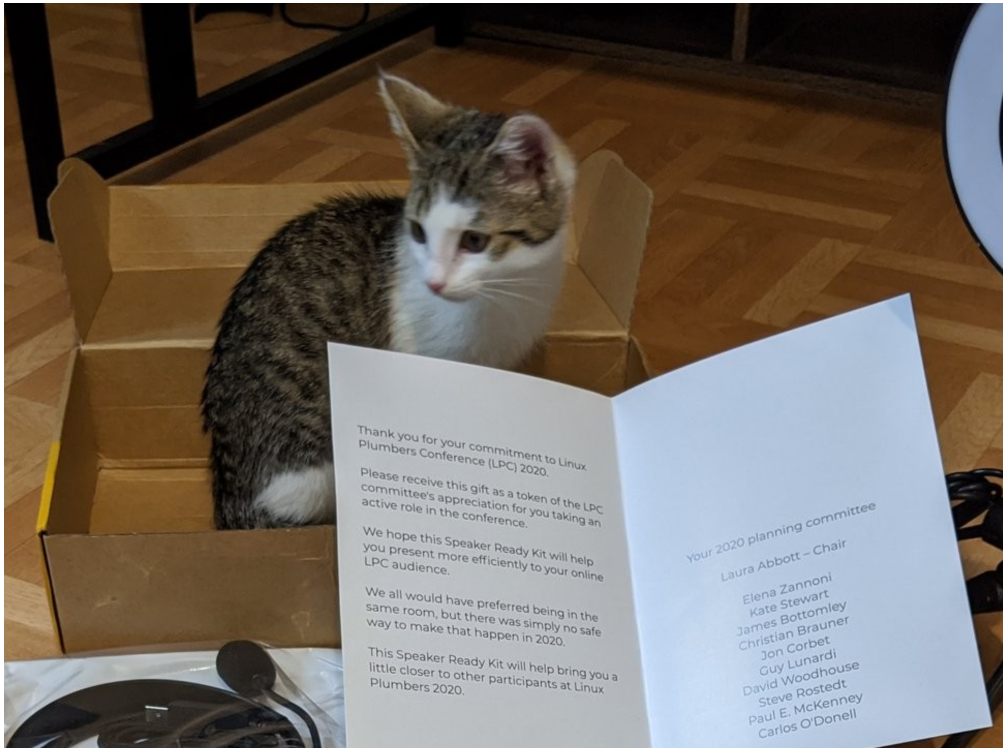
DAMON and DAMOS:

Writing a fine-grained access pattern oriented lightweight kernel module using DAMON/DAMOS in 10 minutes

SeongJae Park <sj@kernel.org>

Disclaimer

- The views expressed herein are those of the speaker;
  they do not reflect the views of his employers
- My cat might come up on the screen. The cat has no '`--silent`' option.
  Sorry, please don't be scared; keep calm and blame COVID19 :P



https://twitter.com/sjpark0x00/status/1295387149018300419/photo/1

I, SeongJae Park <sj@kernel.org>

- Kernel / Hypervisor Engineer at Amazon Web Services
- Interested in the memory management and the parallel programming
- Developing DAMON

This Talk...

- Will not explain how DAMON works internally
  - For that, you can refer to
    - other resources in the project site (https://damonitor.github.io) or
    - the code (https://git.kernel.org/sj/h/damon/next)
- Will explain
  - How, and what kernel hackers (or their kernel subsystems) can get from DAMON (and its not-yet-mainlined features)
  - Things for user-space will not be explained, as this is the Kernel Summit
- Will also discuss about future plans on
  - Extending DAMON for more usages,
  - Improving DAMON itself, and
  - Enhancing MM with DAMON

Overview

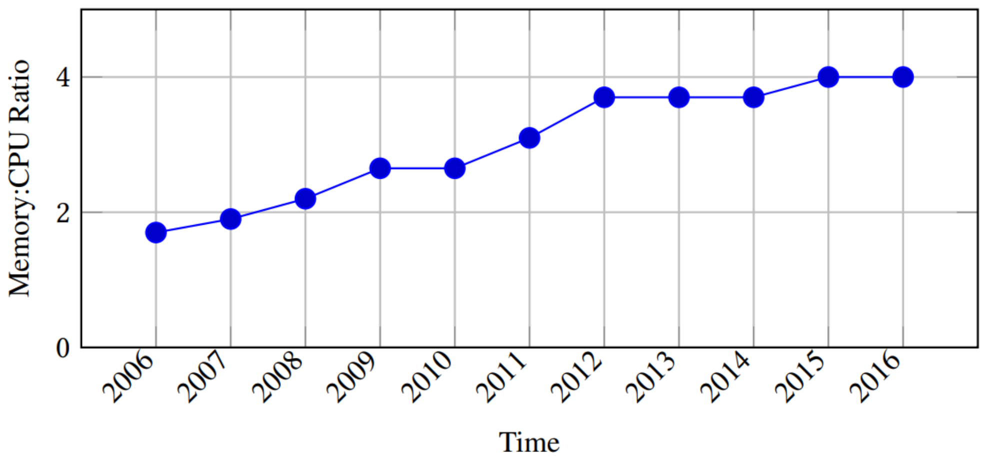
- Motivation
- DAMON
- DAMOS
- DAMON_RECLAIM
- Future Plans
- Summary

Overview

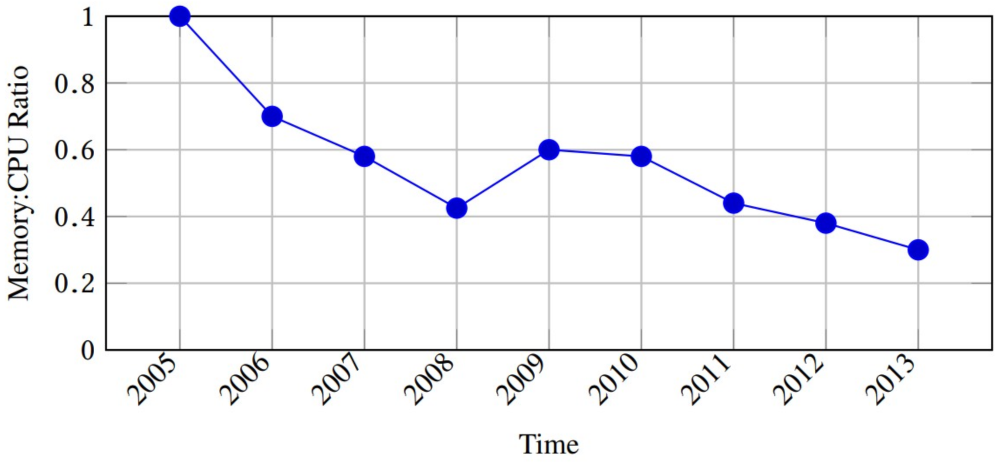
- Motivation
- DAMON
- DAMOS
- DAMON_RECLAIM
- Future Plans
- Summary

Motivation

- Demand for memory is increasing but DRAM supply is not
  - Memory management efficiency is becoming even more important
- Linux MM works with not-so-fine data access information
  - The monitoring overhead is one of the biggest reason



For AWS instances of m* types
(virtual machines: demand)



For multiple server generations
(physical machines: supply)

(Images retrieved from https://oatao.univ-toulouse.fr/24818/1/nitu_24818.pdf)

Overview

- Motivation
- DAMON
  - Intro
  - DAMON Programming Interface
  - Live-coding a Working Set Size Estimation Module
  - DAMON Evaluation
- DAMOS
- DAMON_RECLAIM
- Future Plans
- Summary

DAMON: Data Access MONitor

- A framework for general Data Access MONitoring
  - Provides access frequency of each memory region
  - Allows users practically trade monitoring accuracy for less overhead
    - Provides best-effort accuracy under the condition
    - Users can set upper-bound overhead regardless of the memory size
    - Conceptually scans memory for every 5ms with $< 2\%$ CPU utilization
- The source code is available in
  - Development tree (several not-yet-mainlined features are also here)
  - Back-ports of the development tree for upstream v5.10.y and v5.4.y
  - Amazon Linux kernels (v5.10.y and v5.4.y)
  - The mainline from v5.15-rc1
- A user-space tool and a tests suite are available under GPL v2

How to Use DAMON Programming Interface

- Step 1: Set the requests in ‘`struct damon_ctx`’ instances
  - How, what memory regions of which address spaces should be monitored
  - Where monitoring event notifications should be delivered (callbacks)
    - Users can read the monitoring results or cleanup things inside the function
- Step 2: Start DAMON with the request via ‘`damon_start()`’
  - Then, a kernel thread for the monitoring is created for each request
- Step 3: Do your work in the notification callbacks
  - Monitoring results can be read via ‘`damon_region`’s in the ‘`damon_ctx`’
- Step 4: Finish the monitoring by calling ‘`damon_stop()`’

Live-coding a Working Set Size Estimation Module

- Let’s write a kernel module that
  - Receives pid of a process as a parameter
  - Calculates working set size of the process and log it every 100ms

Live-coding a Working Set Size Estimation Module

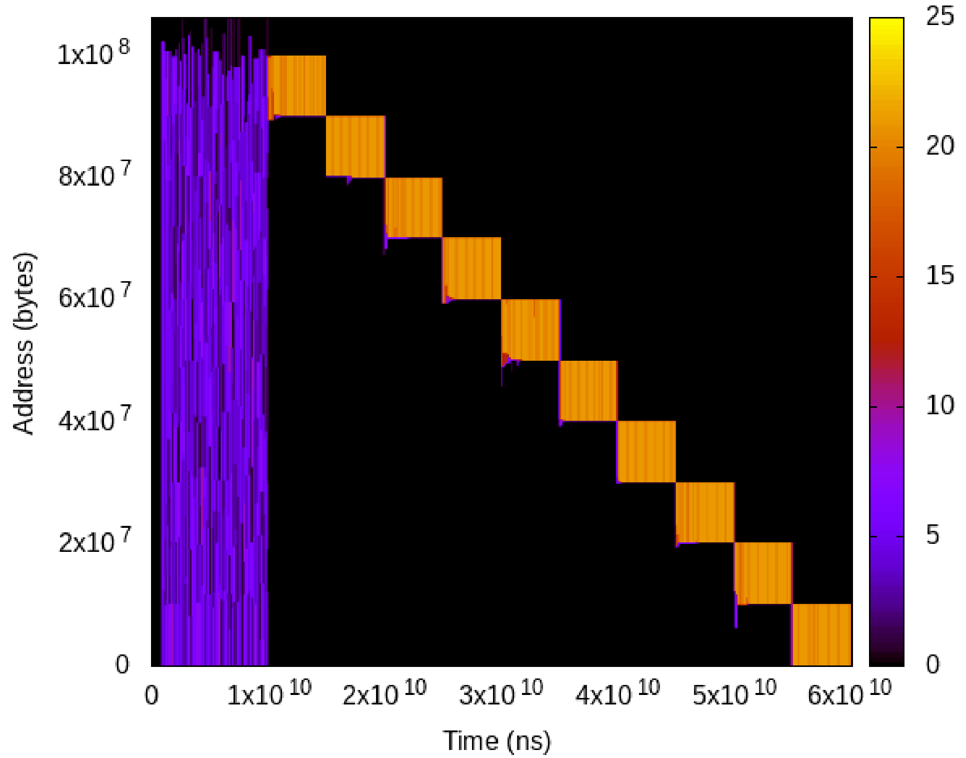
- Let’s write a kernel module that
  - Receives pid of a process as a parameter
  - Calculates working set size of the process and log it every 100ms
  - Live-coded one will be available here
  - Seven lines of code in essence for starting DAMON

```
    /* allocate context */
    ctx = damon_new_ctx(DAMON_ADAPTIVE_TARGET);
2 lines: if (!ctx)--------------------------------------------
    /* specify that we want to monitor virtual address space */
    damon_va_set_primitives(ctx);
    /* specify what process's virtual address space we want to monitor */
    target_pidp = find_get_pid(target_pid);
2 lines: if (!target_pidp)-------------------------------------
    target = damon_new_target((unsigned long)target_pidp);
2 lines: if (!target)------------------------------------------
    damon_add_target(ctx, target);
    /* register callback for reading results */
    ctx->callback.after_aggregation = ksdemo_after_aggregation;
    /* start the monitoring */
    return damon_start(&ctx, 1);
```

Testing The Module

- We will test that against
  - Artificial access pattern generator (‘`$ ./masim ./configs/stairs.cfg`’)
    - Allocates ten 10 MiB objects, accesses all objects for first 10 secs, then accesses the first object for 5 secs, then the second object for 5 secs, …
- We can expect the process will have 100 MiB RSS, while the module reports 10 MiB working set size, after first 10 seconds

Heatmap-format access pattern of the workload. Shows when (x-axis) which memory region (y-axis) is how frequently accessed (color)



Evaluation: How Light-weight DAMON Is?

- For virtual address and physical address monitoring, DAMON...
  - makes the workload 0.62% and 1.53% slower, and
  - Uses 1.76% and 0.96% of single CPU time, respectively
- The overhead is quite low
  - Note: DAMON conceptually scans the memory every 5ms in this case
  - Users can tweak its parameters for less overhead
    - e.g., increasing the memory scan time interval (5ms)

	orig	rec	prec
Runtime (seconds)	191.184	191.563 (+0.62% to orig)	191.928 (+1.53% to orig)
DAMON CPU Usage (%)	0	1.762	0.964

Evaluation: How Light-weight DAMON Is?

- For virtual address and physical address monitoring, DAMON...
  - makes the workload 0.62% and 1.53% slower, and
  - Uses 1.76% and 0.96% of single CPU time, respectively
- The overhead is quite low
  - Note: DAMON conceptually scans the memory every 5ms in this case
  - Users can tweak its parameters for less overhead
    - e.g., increasing the memory scan time interval (5ms)

	orig	rec	prec
Runtime (seconds)	191.184	191.563 (+0.62% to orig)	191.928 (+1.53% to orig)
DAMON CPU Usage (%)	0	1.762	0.964

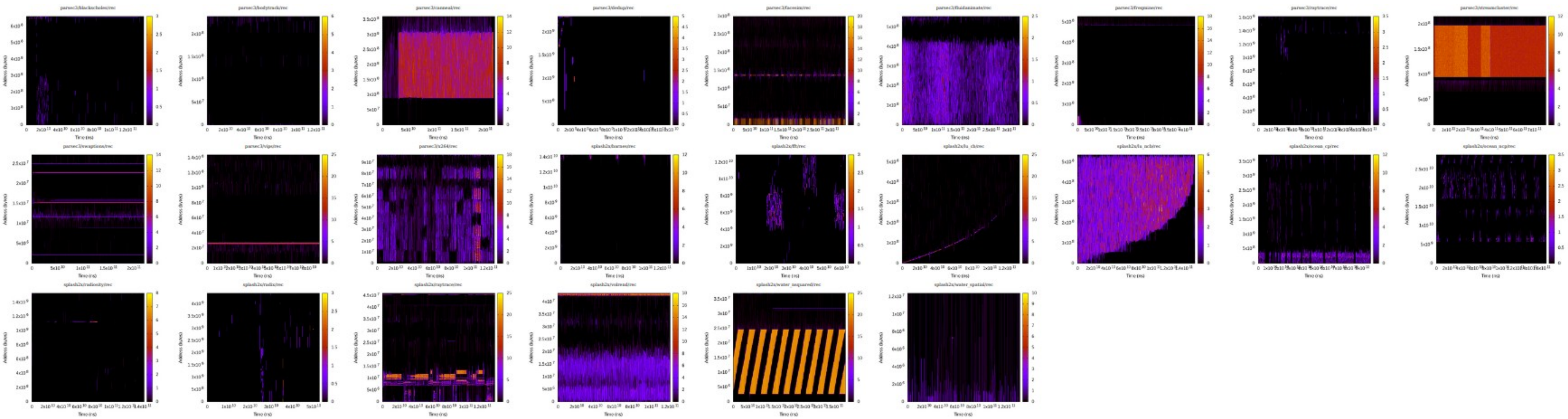
Evaluation: How Light-weight DAMON Is?

- For virtual address and physical address monitoring, DAMON...
  - makes the workload 0.62% and 1.53% slower, and
  - Uses 1.76% and 0.96% of single CPU time, respectively
- The overhead is quite low
  - Note: DAMON conceptually scans the memory every 5ms in this case
  - Users can tweak its parameters for less overhead
    - e.g., increasing the memory scan time interval (5ms)

	orig	rec	prec
Runtime (seconds)	191.184	191.563 (+0.62% to orig)	191.928 (+1.53% to orig)
DAMON CPU Usage (%)	0	1.762	0.964

Evaluation: How Accurate DAMON is?

- No good/easy way for strictly quantize the accuracy, but we can say
  - Visualized monitoring results look reasonable
  - The pattern for ‘masim’ shows expected ones with high accuracy
  - Note that we can adjust the tradeoff for higher accuracy
- More evidence on DAMON accuracy will be introduced in later slides



Overview

- Motivation
- DAMON
- DAMOS
  - Intro
  - DAMOS Programming Interface
  - Live-coding a Proactive Reclamation Kernel Module
  - DAMOS Evaluation
- DAMON_RECLAIM
- Future Plans
- Summary

DAMOS: DAMON-based Operation Schemes

- Imaginable usual DAMON-based MM optimization procedure
  - Monitor data access pattern of some memory range via DAMON,
  - Find regions of interest (e.g., hot or cold) from the results, and
  - Apply some memory management actions to the regions
    - e.g., reclaim cold memory regions, use THP for hot memory regions
- DAMOS is a feature of DAMON; it does above works instead of you
  - Users only need to specify
    - To what specific access pattern (how big, warm, and old) of memory regions
    - What MM action (e.g., reclaim, use THP, ...) they wan to be applied
- Merged in Amazon Linux but mainline, yet
  - Will post the patchset soon

How To Use DAMOS Programming Interface

- Put the monitoring request in ‘`struct damon_ctx`’, as above explained
- Create ‘`struct damos`’ objects and specify the schemes in there
- Specification of each scheme consists with
  - Ranges of size, access frequency, and age of the interest
    - ‘age’ means how long current access pattern has maintained
  - Memory management action that need to be applied to the found regions
- Put the ‘`struct damos`’ objects in the ‘`struct damon_ctx`’ instance
- Then, ‘`damon_start()`’ with the context
  - DAMON starts monitoring as requested in the context, finds the memory regions of the specified pattern, and applies the action

Live-coding a Proactive Reclamation Kernel Module

- Let’s modify the previously written kernel module to
  - Reclaim memory regions of >=4K size that not accessed for >=3 secs

Live-coding a Proactive Reclamation Kernel Module

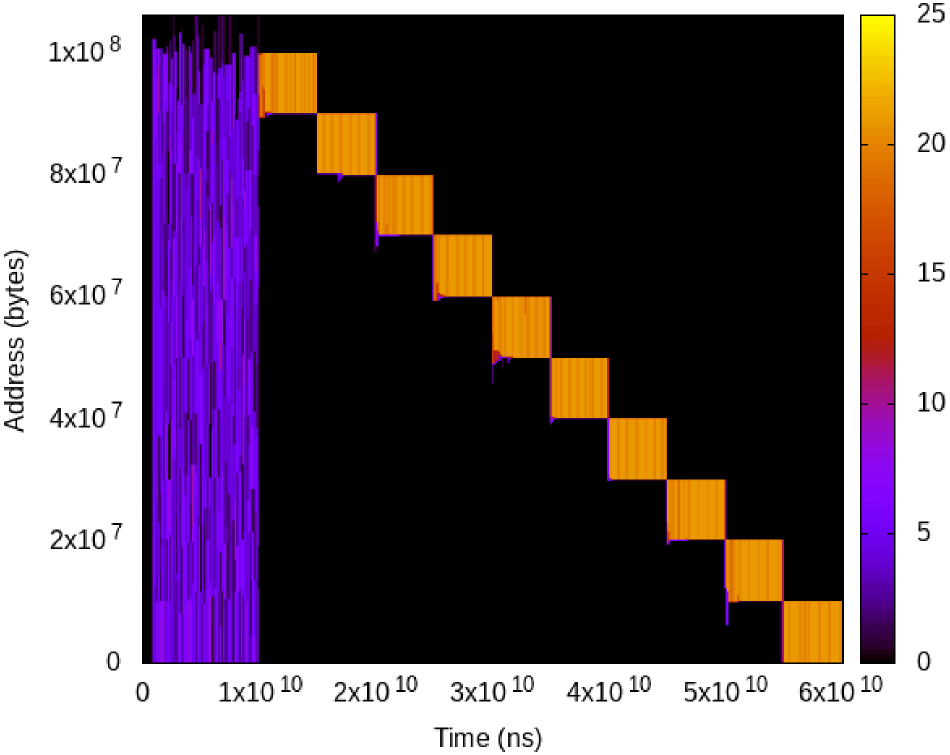
- Let’s modify the previously written kernel module to
  - Reclaim memory regions of >=4K size that not accessed for >=3 secs
  - An example implementation is available here
  - Only two more lines of code in essential

```
        ctx->callback.before_terminate = ksdemo_before_terminate;

        /* create the operation scheme specficiation */
        scheme = damon_new_scheme(
                        /* Find regions having size >= PAGE_SIZE */
                        PAGE_SIZE, ULONG_MAX,
                        /* and not accessed at all */
                        0, 0,
                        /* for 50 aggregation interval (5 secs). */
                        30, UINT_MAX,
                        /* Then, page out those as soon as found */
                        DAMOS_PAGEOUT,
                        &quota, &wmarks);
+--  4 lines: if (!scheme) {-----------------------------------------
        /* register the scheme */
        err = damon_set_schemes(ctx, &scheme, 1);
+--  2 lines: if (err)-----------------------------------------------

        err = damon_start(&ctx, 1);
```

Testing The Proactive Reclamation Module

- We will test that against the stairs access pattern, again
  - Allocates ten 10 MiB objects, accesses all for first 10 secs, then accesses the first object for 5 secs, then the second object for 5 secs, …
- The module is expected to
  - Shrink the process's RSS to 10 MiB after the first 13 seconds

Heatmap-format access pattern of the workload. Shows when (x-axis) which memory region (y-axis) is how frequently accessed (color)



Example Schemes For Evaluation of DAMOS

- ethp: Enhanced THP
  - `MADV_THP` for memory regions that real access is monitored
  - `MADV_NOTHP` for >=2MB memory regions that not accessed >=7 secs
  - Expected to reduce THP's internal fragmentation caused memory bloats

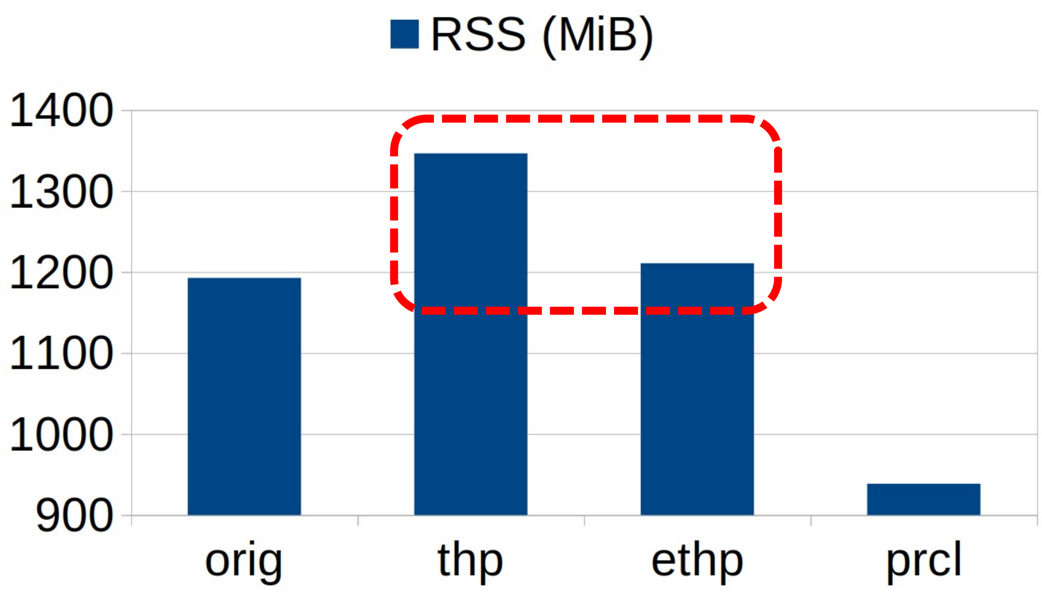
```
$ cat ethp.damos
# for regions having 5/100 access frequency, apply MADV_HUGEPAGE
min max     5 max       min max     hugepage
# for regions >=2MB and not accessed for >=7 seconds, apply MADV_NOHUGEPAGE
2M max      min min     7s max      nohugepage
```

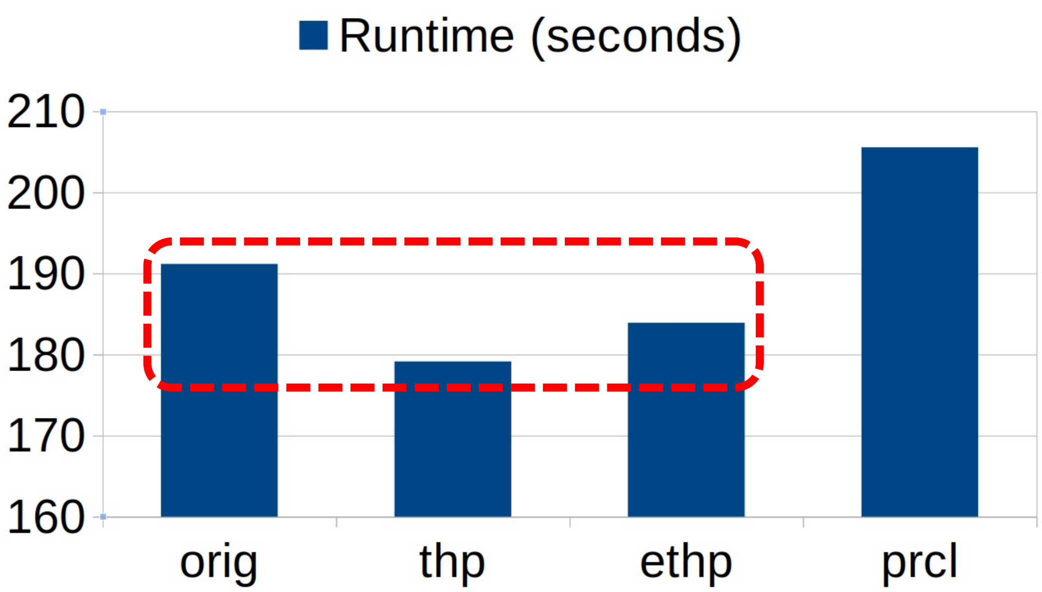
- prcl: Proactive Reclamation
  - Reclaim memory regions that not accessed >= 10secs
  - Expected to reduce memory footage with minimal performance drops

```
$ cat prcl.damos
# for regions >=4KB and not accessed for >=10 seconds, apply MADV_PAGEOUT
4K  max     0 0     10s max     pageout
```

How Effective DAMOS Is? (How Accurate DAMON Is?)

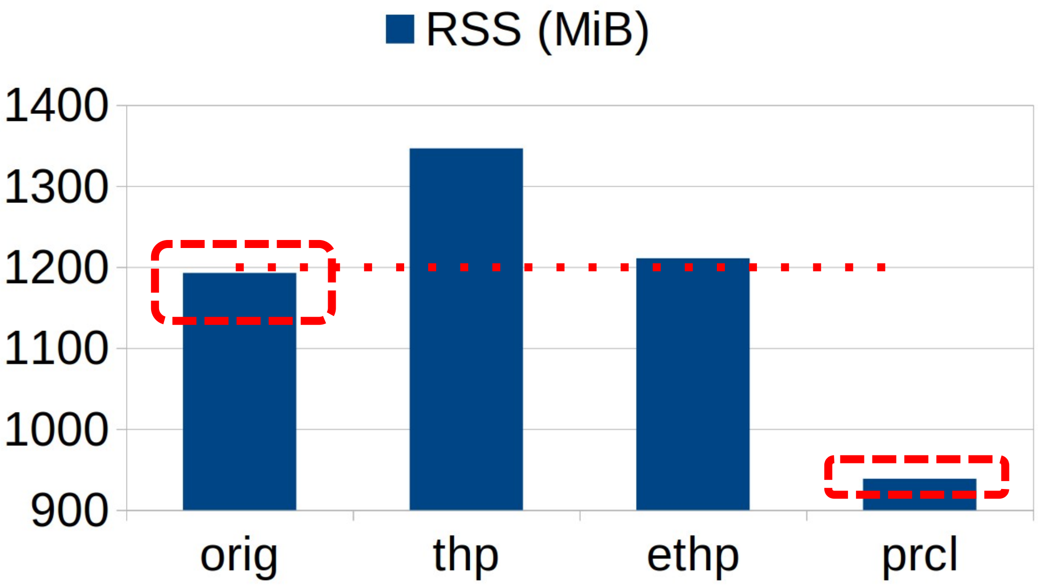
- ‘ethp’ reduces 76% of ‘thp’ (‘always’ THP policy) memory overhead while preserving 25% of ‘thp’ performance improvement
- ‘prcl’ saves 38.46% memory with 8.26% runtime slowdown
- Working as expected and seems effective (DAMON is accurate)
- But… 8.26% slowdown?

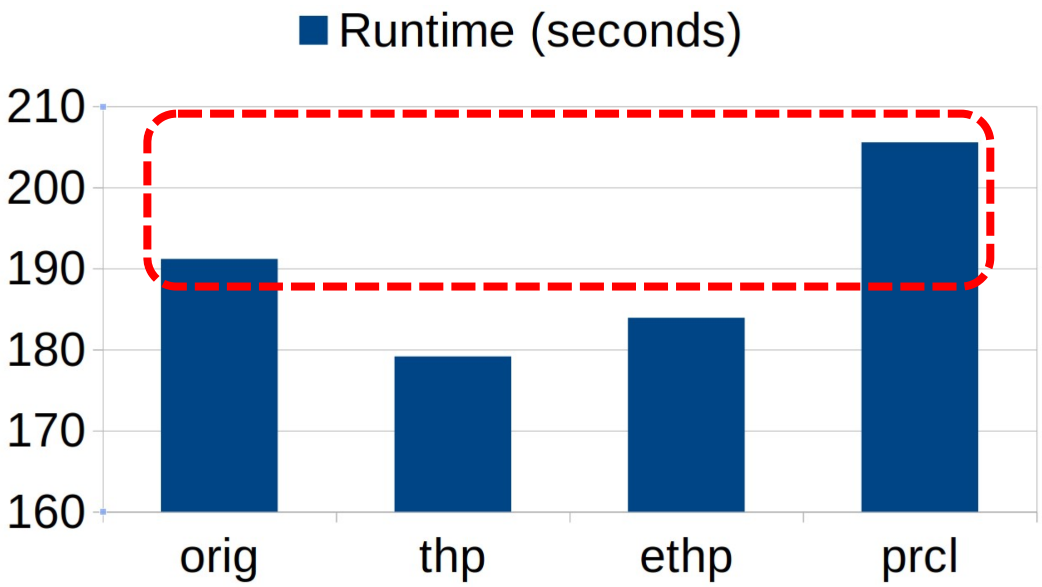




How Effective DAMOS Is? (How Accurate DAMON Is?)

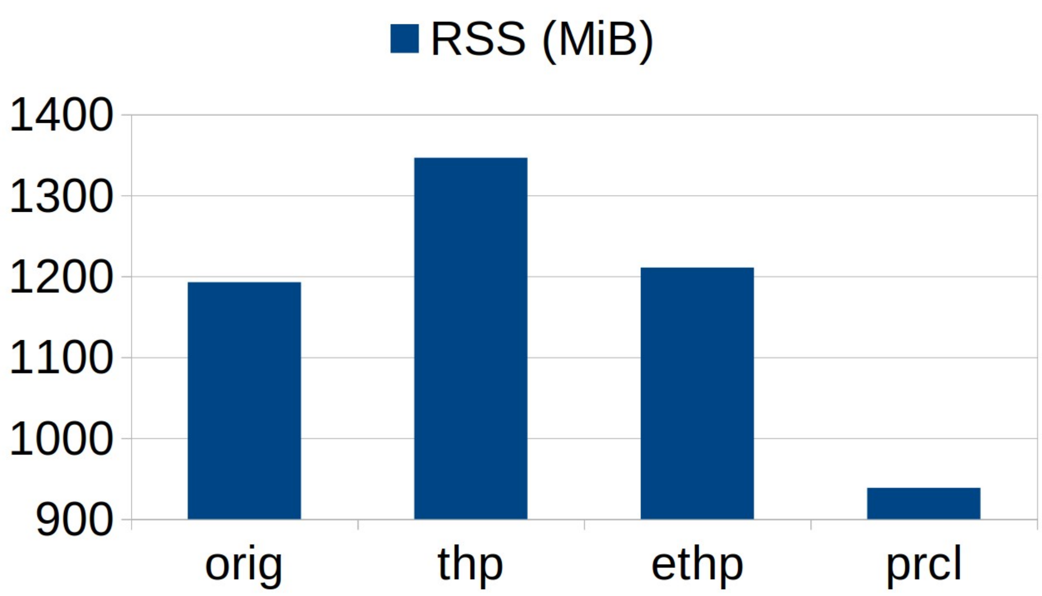
- ‘ethp’ reduces 76% of ‘thp’ (‘always’ THP policy) memory overhead while preserving 25% of ‘thp’ performance improvement
- ‘prcl’ saves 38.46% memory with 8.26% runtime slowdown
- Working as expected and seems effective (DAMON is accurate)
- But… 8.26% slowdown?

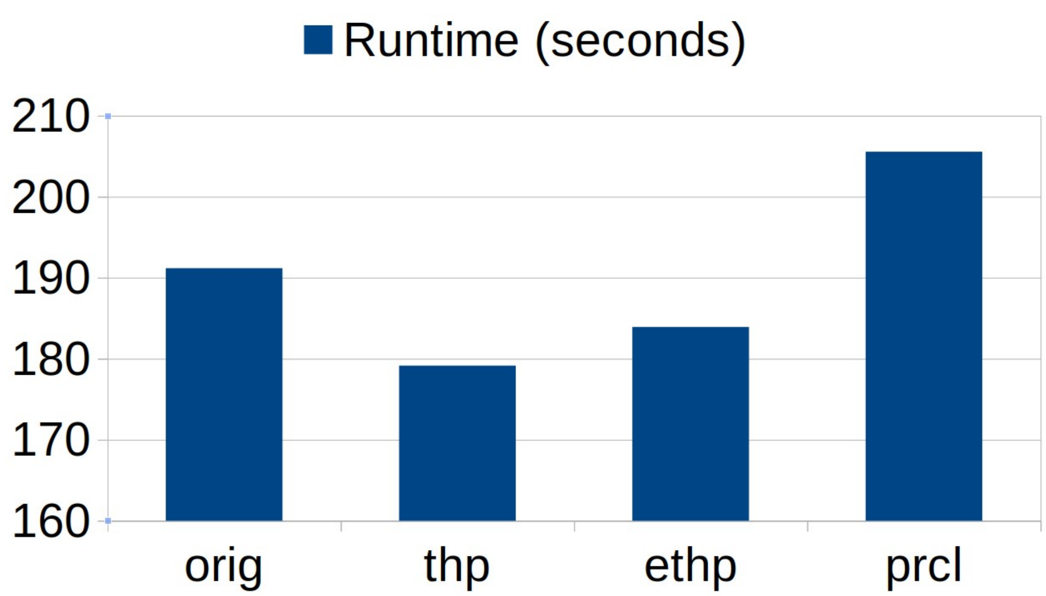




How Effective DAMOS Is? (How Accurate DAMON Is?)

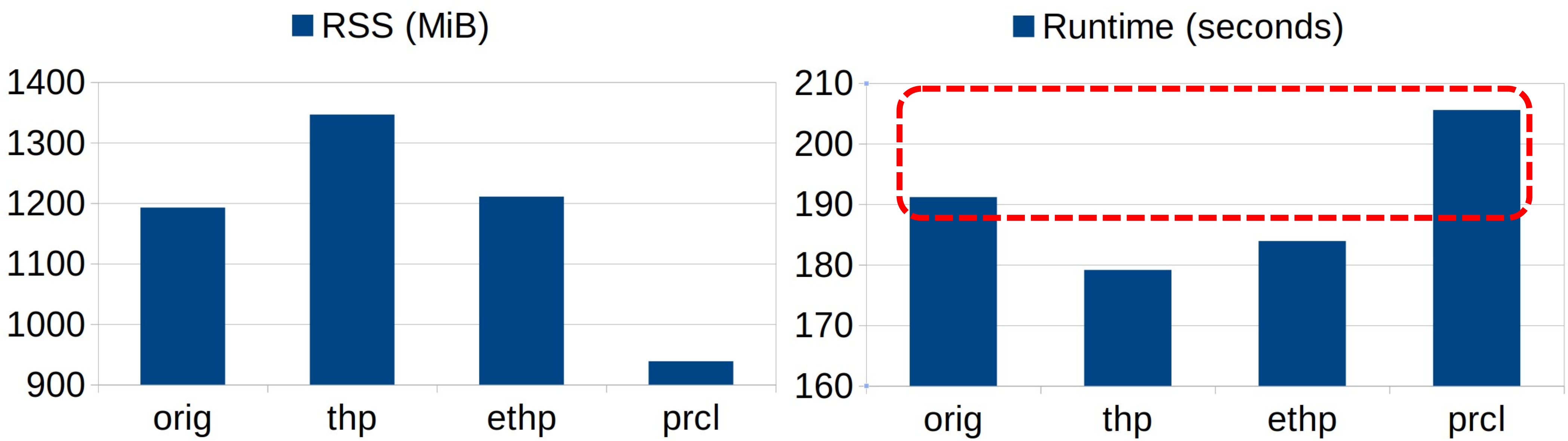
- ‘ethp’ reduces 76% of ‘thp’ (‘always’ THP policy) memory overhead while preserving 25% of ‘thp’ performance improvement
- ‘prcl’ saves 38.46% memory with 8.26% runtime slowdown
- Working as expected and seems effective (DAMON is accurate)
- But… 8.26% slowdown?





How Effective DAMOS Is? (How Accurate DAMON Is?)

- ‘ethp’ reduces 76% of ‘thp’ (‘always’ THP policy) memory overhead while preserving 25% of ‘thp’ performance improvement
- ‘prcl’ saves 38.46% memory with 8.26% runtime slowdown
- Working as expected and seems effective (DAMON is accurate)
- But… 8.26% slowdown?



DAMOS Challenges for Production Usage

- 8.26% slowdown of ‘prcl’ seems too huge for the production
  - Might be reasonable depending on the specific requirement, though
  - Can mitigate by tuning the scheme to be less aggressive
- DAMOS schemes tuning is challenging
  - Tuning is needed for for each workload and system
  - The thresholds are not intuitive for sysadmins
- Auto-tuning programs can be a solution
  - Our simple auto-tuner makes ‘prcl’ achieve
    - 24.97% memory saving with 0.91% runtime slowdown
    - (Untuned PRCL: 38.46% memory saving with 8.26% runtime slowdown)
- But, couldn’t the kernel just work without such user-space help?

Overview

- Motivation
- DAMON
- DAMOS
- DAMON_RECLAIM
  - DAMOS Safety Guarantees
  - DAMON_RECLAIM Intro
- Future Plans
- Summary

DAMOS Safety Guarantees

- For productions that prefer safety, DAMOS provides additional features
- Time/space quota per a given time interval
  - DAMOS uses CPU time no more than the given time quota
  - DAMOS applies the action to memory no more than the space quota
- Regions prioritization
  - Under the quota, DAMOS applies the action to prioritized regions first
  - Prioritization logic can be customized for different DAMOS actions
    - In case of RECLAIM, older and colder pages are prioritized by default
- Three watermarks (high, mid, low) with user-specified metric (e.g., freemem)
  - Deactivate if the metric > high_watermark or metric < low_watermark
  - Activate if the metric < mid_watermark and metric > low_watermark
  - Avoid DAMOS using any resource under a peaceful or a catastrophic situation

Evaluation of DAMOS Safety Guarantees

- ‘prcl’ for the physical address space with different safety guarantees
- Smaller time quota reduces DAMON’s CPU usage and slowdown
  - Note that it also reduces the memory saving, as being less aggressive
- Enabling prioritization further reduces slowdown
- Still need tuning, but the knobs would be intuitive for sysadmins

time quota	prioritization	memory_saving	cpu_util	slowdown
N	N	47.16%	11.62%	5.40%
200ms/s	N	48.42%	10.92%	4.69%
50ms/s	N	40.84%	5.70%	4.53%
10ms/s	N	4.55%	1.78%	2.51%
200ms/s	Y	47.99%	10.41%	5.10%
50ms/s	Y	40.34%	5.16%	3.38%
10ms/s	Y	0.77%	1.37%	1.84%

Evaluation of DAMOS Safety Guarantees

- ‘prcl’ for the physical address space with different safety guarantees
- Smaller time quota reduces DAMON’s CPU usage and slowdown
  - Note that it also reduces the memory saving, as being less aggressive
- Enabling prioritization further reduces slowdown
- Still need tuning, but the knobs would be intuitive for sysadmins

time quota	prioritization	memory_saving	cpu_util	slowdown
N	N	47.16%	11.62%	5.40%
200ms/s	N	48.42%	10.92%	4.69%
50ms/s	N	40.84%	5.70%	4.53%
10ms/s	N	4.55%	1.78%	2.51%
200ms/s	Y	47.99%	10.41%	5.10%
50ms/s	Y	40.34%	5.16%	3.38%
10ms/s	Y	0.77%	1.37%	1.84%

Evaluation of DAMOS Safety Guarantees

- ‘prcl’ for the physical address space with different safety guarantees
- Smaller time quota reduces DAMON’s CPU usage and slowdown
  - Note that it also reduces the memory saving, as being less aggressive
- Enabling prioritization further reduces slowdown
- Still need tuning, but the knobs would be intuitive for sysadmins

time quota	prioritization	memory_saving	cpu_util	slowdown
N	N	47.16%	11.62%	5.40%
200ms/s	N	48.42%	10.92%	4.69%
50ms/s	N	40.84%	5.70%	4.53%
10ms/s	N	4.55%	1.78%	2.51%
200ms/s	Y	47.99%	10.41%	5.10%
50ms/s	Y	40.34%	5.16%	3.38%
10ms/s	Y	0.77%	1.37%	1.84%

DAMON_RECLAIM

- DAMON-based proactive reclamation kernel module
- Written using DAMOS
  - Excepting the code for module parameters, only 188 lines of code
- Aims to be used on production
  - Ensure the safety using the quotas and watermarks
  - The quotas ans watermarks can be tweaked via module parameters

Overview

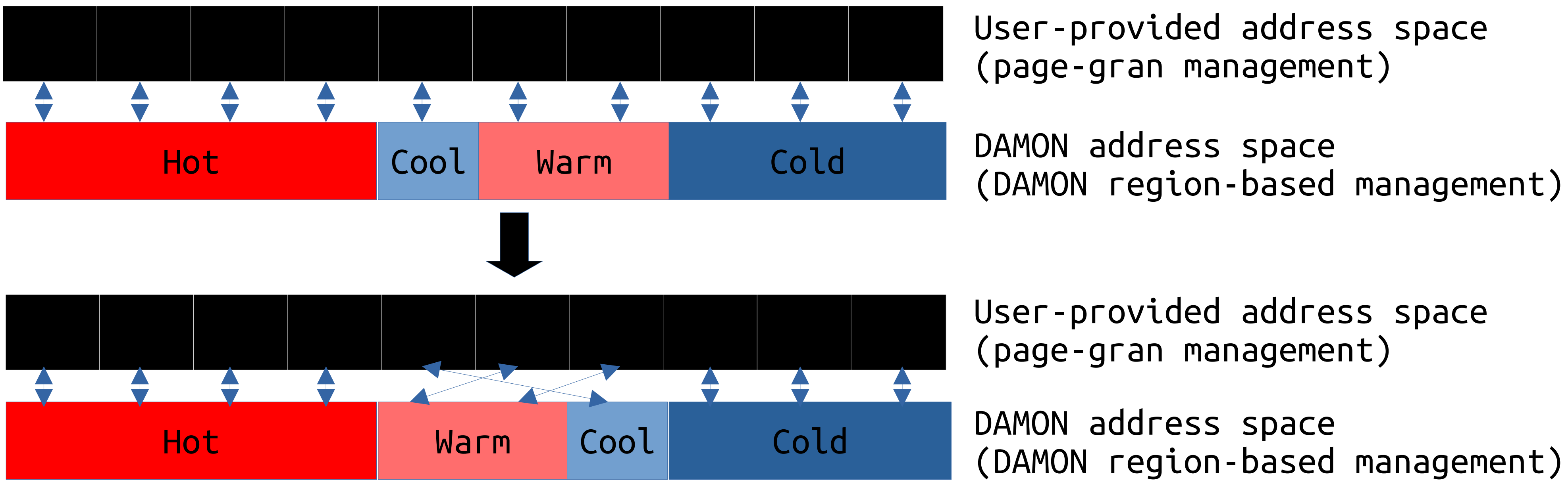
- Motivation
- DAMON
- DAMOS
- DAMON_RECLAIM
- Future Plans
  - Extending DAMON
  - Improving DAMON
  - Improving MM with DAMON
- Summary

Extending DAMON (only brainstorming)

- DAMON can be extended for various address spaces and use cases
  - Need to implement new monitoring primitives for the use case
- Currently, monitoring primitives for only virtual address spaces, the physical address space, and page-granularity system monitoring are available
- Imaginable extensions include
  - More efficient page-granularity system monitoring
    - Current page-granularity monitoring is only for proof of concepts
    - MGLRU’s page table-based scanning might be able to be used for this
  - for specific cgroups,
  - for only specific file-backed memory,
  - for read-only or write-only

Improving DAMON (only brainstorming)

- DAMON's accuracy and overhead could be more optimized
  - Adaptive monitoring attributes adjustment and regions splitting
    - Find too stable or too unstable regions and do more aggressive monitoring
  - Remapping regions based on monitoring results, to sorted by hotness
    - The spatial locality assumption of memory regions will be more reasonable
    - DAMON-internal address space would be needed for usual cases



Improving MM with DAMON (only brainstorming)

- DAMON might be able to be used to help
  - THP promotion/demotion
  - Page migration target (for compaction or CMA) selection
  - LRU pages prioritization
  - Tiered-memory management
- The works could fundamentally be done in two ways
  - Implementing new subsystems
  - Modifying existing subsystems
  - Any opinion or preference among these?
    - I guess it should be depend on each specific case, though…

Overview

- Motivation
- DAMON
- DAMOS
- DAMON_RECLAIM
- Future Plans
- Summary

Summary

- DAMON/DAMOS helps you write fine-grained data access pattern-oriented light-weight kernel modules
- Such modules could be useful for enhancing memory efficiency
- There are many more things to do; Looking forward your contributions
- For more information
  - please visit https://damonitor.github.io, or
  - reach out to sj@kernel.org

Special Thanks To (Alphabetical order)

- I might missed someone's name, please forgive me...

Alexander Shishkin
Amit Shah
Andrew Morton
Brendan Higgins
David Hildenbrand
David Rientjes
David Woodhouse
Fan Du
Fernand Sieber
Greg Kroah-Hartman
Greg Thelen
Jonathan Cameron
Jonathan Corbet
Leonard Foerster
Marco Elver
Markus Boehme
Maximilian Heyne
Minchan Kim
Paul E. McKenney
Shakeel Butt
Stefan Nuernberger
Steven Rostedt
Varad Gautam
Yunjae Lee

Questions?

- You can also
  - visit **https://damonitor.github.io**, or
  - reach out to **sj@kernel.org**

https://kids.nationalgeographic.com/content/dam/kids/photos/animals/Birds/A-G/adelie-penguin-jumping-ocean.ngsversion.1396530997321.adapt.1900.1.jpg

Backup Slides

boilerplate

```
// SPDX-License-Identifier: GPL-2.0

#define pr_fmt(fmt) "ksdemo: " fmt

#include <linux/init.h>
#include <linux/kernel.h>
#include <linux/module.h>

static int __init ksdemo_init(void)
{
        pr_info("Hello Kernel Summit 2021\n");
        return 0;
}

static void __exit ksdemo_exit(void)
{
        pr_info("Goodbye Kernel Summit 2021\n");
}

module_init(ksdemo_init);
module_exit(ksdemo_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("SeongJae Park");
MODULE_DESCRIPTION("Kernel Summit 2021 live coding demo");
```

diff -u boilerplate wsse (1/4)

```
@@ -2,18 +2,69 @@

 #define pr_fmt(fmt) "ksdemo: " fmt

+#include <linux/damon.h>
 #include <linux/init.h>
 #include <linux/kernel.h>
 #include <linux/module.h>
+#include <linux/pid.h>
+
+static int target_pid __read_mostly;
+module_param(target_pid, int, 0600);
+
+struct damon_ctx *ctx;
+struct pid *target_pidp;
[...]
```

diff -u boilerplate wsse (2/4)

```
[...]
+
+static int ksdemo_after_aggregation(struct damon_ctx *c)
+{
+	struct damon_target *t;
+
+	damon_for_each_target(t, c) {
+		struct damon_region *r;
+		unsigned long wss = 0;
+
+		damon_for_each_region(r, t) {
+			if (r->nr_accesses > 0)
+				wss += r->ar.end - r->ar.start;
+		}
+		pr_info("wss: %lu\n", wss);
+	}
+	return 0;
+}
[...]
```

diff -u boilerplate wsse (3/4)

```
[...]
 static int __init ksdemo_init(void)
 {
+	struct damon_target *target;
+
	pr_info("Hello Kernel Summit 2021\n");
-	return 0;
+
+	/* allocate context */
+	ctx = damon_new_ctx(DAMON_ADAPTIVE_TARGET);
+	if (!ctx)
+		return -ENOMEM;
+	/* specify that we want to monitor virtual address space */
+	damon_va_set_primitives(ctx);
+	/* specify what process's virtual address space we want to monitor */
+	target_pidp = find_get_pid(target_pid);
+	if (!target_pidp)
+		return -EINVAL;
+	target = damon_new_target((unsigned long)target_pidp);
+	if (!target)
+		return -ENOMEM;
+	damon_add_target(ctx, target);
+	/* register callback for reading results */
+	ctx->callback.after_aggregation = ksdemo_after_aggregation;
+	/* start the monitoring */
+	return damon_start(&ctx, 1);
 }
[...]
```

diff -u boilerplate wsse (4/4)

```
[…]

 static void __exit ksdemo_exit(void)
 {
+       if (ctx) {
+               damon_stop(&ctx, 1);
+               damon_destroy_ctx(ctx);
+       }
+       if (target_pidp)
+               put_pid(target_pidp);
        pr_info("Goodbye Kernel Summit 2021\n");
 }
```

diff -u wsse prcl (1/2)

```
@@ -34,6 +34,9 @@
 static int __init ksdemo_init(void)
 {
        struct damon_target *target;
+       struct damos *scheme;
+       struct damos_quota quota = {};
+       struct damos_watermarks wmarks = {};

        pr_info("Hello Kernel Summit 2021\n");

[...]
```

diff -u wsse prcl (2/2)

```
[...]
@@ -53,6 +56,22 @@
        damon_add_target(ctx, target);
        /* register callback for reading results */
        ctx->callback.after_aggregation = ksdemo_after_aggregation;
+
+       /* create the operation scheme specification */
+       scheme = damon_new_scheme(
+                       /* find regions having size >= PAGE_SIZE */
+                       PAGE_SIZE, ULONG_MAX,
+                       /* and not accessed at all */
+                       0, 0,
+                       /* for 30 aggregation interval (3 secs) */
+                       30, UINT_MAX,
+                       /* and page out those */
+                       DAMOS_PAGEOUT,
+                       &quota, &wmarks);
+       if (!scheme)
+               return -ENOMEM;
+       damon_set_schemes(ctx, &scheme, 1);
+
        /* start the monitoring */
        return damon_start(&ctx, 1);
 }
```

Evaluation Environment

- Test machine
  - QEMU/KVM virtual machine on AWS EC2 i3.metal instance
  - 36 vCPUs, 128 GB memory, 4 GB zram swap device
  - Ubuntu 18.04, THP enabled policy `madvise`
  - Linux v5.15-rc1 based DAMON dev tree (The source tree is available)
- Workloads: 25 realistic benchmark workloads
  - 13 workloads from PARSEC3
  - 12 workloads from SPLASH-2X
- DAMON monitoring attributes: The default values
  - 5ms sampling, 100ms aggregation, and 1s regions update intervals
  - Number of regions: [10, 1000]

Evaluation Setup: DAMON

- Questions to Answer
  - How lightweight DAMON is?
  - How accurate DAMON is?
- Run 25 workloads from PARSEC3 and SPLASH-2X one by one on three different systems
  - orig: v5.15-rc1, thp for only ‘madvise’
  - rec: orig + DAMON running for the workload’s virtual address space
  - prec: orig + DAMON running for the entire physical address space
- Measure the workload’s runtime and DAMON’s CPU usage
- For more details in the setup, refer to backup slides

Evaluation Setup: DAMOS

- Questions to answer
  - How effective DAMOS is?
    - This also answers ‘How accurate DAMON is?’
- Basically similar to that for DAMON
  - Run the 25 workloads and measure some metrics
  - Apply some DAMON-based operation schemes to the workloads