← Back

maple_tree: move mas_spanning_rebalance loop to function

#Reviewed

Author: Liam R. Howlett <Liam.Howlett@oracle.com>
Date: 3/27/2026, 6:30:25 PM
Subsystems: -
Patches: 29 / 29
Model: gemini/gemini-3.1-pro-preview
Sashiko ver.: bc895fbc

AI Reviews

Patch	Potential Regressions
Patch 1: maple_tree: move mas_spanning_rebalance loop to function	✓
Patch 2: maple_tree: extract use of big node from mas_wr_spanning_store()	✓
Patch 3: maple_tree: remove unnecessary assignment of orig_l index	✓
Patch 4: maple_tree: inline mas_spanning_rebalance() into mas_wr_spanning_rebalance()	✓
Patch 5: maple_tree: make ma_wr_states reliable for reuse in spanning store	0 0 1 0
Patch 6: maple_tree: remove l_wr_mas from mas_wr_spanning_rebalance	✓
Patch 7: maple_tree: don't pass through height in mas_wr_spanning_store	✓
Patch 8: maple_tree: move maple_subtree_state from mas_wr_spanning_store to mas_wr_spanning_rebalance	✓
Patch 9: maple_tree: correct right ma_wr_state end pivot in mas_wr_spanning_store()	0 0 0 1
Patch 10: maple_tree: introduce maple_copy node and use it in mas_spanning_rebalance()	✓
Patch 11: maple_tree: testing update for spanning store	0 0 2 2
Patch 12: maple_tree: inline mas_spanning_rebalance_loop() into mas_wr_spanning_rebalance()	✓
Patch 13: maple_tree: change initial big node setup in mas_wr_spanning_rebalance()	✓

/ search j k nav Enter open q back

Patch	Potential Regressions
Patch 14: maple_tree: introduce ma_leaf_max_gap()	✓
Patch 15: maple_tree: add gap support, slot and pivot sizes for maple copy	✓
Patch 16: maple_tree: start using maple copy node for destination	**1 2 0 0**
Patch 17: maple_tree: inline mas_wr_spanning_rebalance()	✓
Patch 18: maple_tree: remove unnecessary return statements	✓
Patch 19: maple_tree: separate wr_split_store and wr_rebalance store type code path	**0 0 1 0**
Patch 20: maple_tree: add cp_is_new_root() helper	✓
Patch 21: maple_tree: use maple copy node for mas_wr_rebalance() operation	**1 0 0 0**
Patch 22: maple_tree: add test for rebalance calculation off-by-one	**0 0 1 0**
Patch 23: maple_tree: add copy_tree_location() helper	✓
Patch 24: maple_tree: add cp_converged() helper	✓
Patch 25: maple_tree: use maple copy node for mas_wr_split()	✓
Patch 26: maple_tree: remove maple big node and subtree structs	✓
Patch 27: maple_tree: pass maple copy node to mas_wmb_replace()	✓
Patch 28: maple_tree: don't pass end to mas_wr_append()	✓
Patch 29: maple_tree: clean up mas_wr_node_store()	✓

Patch 1: maple_tree: move mas_spanning_rebalance loop to function

Result: **No regressions** **# Reviewed**

▶ **Inline Review**

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 192,759 · Cached: 986,461 · Out: 827

View Raw Log ↑ Back to Summary

Patch 2: maple_tree: extract use of big node from mas_wr_spanning_store()

Result: **No regressions** **# Reviewed**

▶ **Inline Review**

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 187,888 · Cached: 1,117,748 · Out: 1,194

View Raw Log ↑ Back to Summary

/ search j k nav Enter open g back

Patch 3: maple_tree: remove unnecessary assignment of orig_l index

Result: No regressions # Reviewed

▶ Inline Review

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 727,687 · Cached: 4,721,240 · Out: 5,774

View Raw Log ↑ Back to Summary

Patch 4: maple_tree: inline mas_spanning_rebalance() into mas_wr_spanning_rebalance()

Result: No regressions # Reviewed

▶ Inline Review

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 569,378 · Cached: 5,532,135 · Out: 4,481

View Raw Log ↑ Back to Summary

Patch 5: maple_tree: make ma_wr_states reliable for reuse in spanning store

Result: Critical: 0 · High: 0 · Medium: 1 · Low: 0 # Reviewed

▼ Inline Review

Copy

```
commit 1f11317e78864ffbf49156d9e0b1e32173a0cfdd
Author: Liam R. Howlett <Liam.Howlett@oracle.com>

maple_tree: make ma_wr_states reliable for reuse in spanning store

This modifies mas_extend_spanning_null() to update the range min and max
of the resulting store operation. This ensures the maple write state
matches the operation for future changes.

Link: https://lkml.kernel.org/r/20260130205935.2559335-7-Liam.Howlett@oracle.com

> diff --git a/lib/maple_tree.c b/lib/maple_tree.c
> index fb14ce4a49c3..ab14876bebf7 100644
> --- a/lib/maple_tree.c
> +++ b/lib/maple_tree.c
> @@ -3319,6 +3319,7 @@ static inline void mas_extend_spanning_null(struct ma_wr_state
*l_wr_mas,
>                         l_mas->index = l_mas->min;
>
>                 l_mas->offset = l_slot - 1;
> +               l_wr_mas->r_min = l_mas->index;
>         }
>
>         if (!r_wr_mas->content) {

[ ... ]

> @@ -3331,6 +3332,7 @@ static inline void mas_extend_spanning_null(struct ma_wr_state
*l_wr_mas,
>                 r_mas->last = mas_safe_pivot(r_mas, r_wr_mas->pivots,
>                                              r_wr_mas->type, r_mas->offset + 1);
```

/ search j k nav Enter open q back

```
>                r_mas->offset++;
> +              r_wr_mas->r_max = r_mas->last;
```

> Is it possible that this block is mathematically unreachable, preventing
> r_wr_mas->r_max from being updated?
>
> On the left side of mas_extend_spanning_null(), two independent if
> statements are used, but the right side uses an else if block.
>
> If !r_wr_mas->content is false, the code evaluates the condition:
> (r_mas->last == r_wr_mas->r_max) && (r_mas->last < r_mas->max)
>
> Since r_wr_mas is initialized by searching for mas->last + 1,
> r_wr_mas->r_max is guaranteed to be greater than mas->last. This makes
> r_mas->last == r_wr_mas->r_max strictly false.
>
> Even in the overflow case where mas->last == ULONG_MAX, the subsequent
> condition r_mas->last < r_mas->max becomes ULONG_MAX < r_mas->max,
> which is also false.
>
> Could this mean the else if block is dead code, and the new update to
> r_wr_mas->r_max will never execute?

```
>        }
>  }
```

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 283,897 · Cached: 2,163,219 · Out: 3,856

View Raw Log ↑ Back to Summary

Patch 6: maple_tree: remove l_wr_mas from mas_wr_spanning_rebalance

Result: **No regressions** **# Reviewed**

▶ **Inline Review**

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 1,826,266 · Cached: 12,242,362 · Out: 8,994

View Raw Log ↑ Back to Summary

Patch 7: maple_tree: don't pass through height in mas_wr_spanning_store

Result: **No regressions** **# Reviewed**

▶ **Inline Review**

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 357,192 · Cached: 3,340,371 · Out: 2,444

View Raw Log ↑ Back to Summary

Patch 8: maple_tree: move maple_subtree_state from mas_wr_spanning_store to mas_wr_spanning_rebalance

Result: **No regressions** / search j k nav Enter open r back **# Reviewed**

▶ **Inline Review**

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 746,706 · Cached: 4,657,209 · Out: 2,754

View Raw Log ↑ Back to Summary

Patch 9: maple_tree: correct right ma_wr_state end pivot in mas_wr_spanning_store()

Result: Critical: 0 · High: 0 · Medium: 0 · Low: 1 **# Reviewed**

▼ **Inline Review**

Copy

```
commit c2f3fbf90c887555be8c45da38a613213f78328f
Author: Liam R. Howlett <Liam.Howlett@oracle.com>

maple_tree: correct right ma_wr_state end pivot in mas_wr_spanning_store()

This commit sets r_wr_mas.end_piv to r_wr_mas.r_max early in
mas_wr_spanning_store() to prepare the write state for upcoming patches.

Link: https://lkml.kernel.org/r/20260130205935.2559335-11-Liam.Howlett@oracle.com
```

```
> diff --git a/lib/maple_tree.c b/lib/maple_tree.c
> index c5bb341da5e9..caac936bd8d4 100644
> --- a/lib/maple_tree.c
> +++ b/lib/maple_tree.c
> @@ -3494,6 +3494,7 @@ static void mas_wr_spanning_store(struct ma_wr_state *wr_mas)
>       r_mas.index = r_mas.last;
>       mas_wr_walk_index(&r_wr_mas);
>       r_mas.last = r_mas.index = mas->last;
> +     r_wr_mas.end_piv = r_wr_mas.r_max;
>
>       /* Set up left side. */
>       mas_wr_walk_index(wr_mas);
```

```
If the store is a NULL store, mas_extend_spanning_null() is called
shortly after this assignment:

        if (!wr_mas->entry) {
                mas_extend_spanning_null(wr_mas, &r_wr_mas);
                mas->last = r_mas.last;
        }

Does setting r_wr_mas.end_piv before this extension cause end_piv to
become stale?

Looking at mas_extend_spanning_null(), r_wr_mas.r_max can be updated
to the new boundary by absorbing adjacent NULL slots:

        if (!r_wr_mas->content) {
                ...
        } else if ((r_mas->last == r_wr_mas->r_max) &&
            (r_mas->last < r_mas->max) &&
            !mas_slot_locked(r_mas, r_wr_mas->slots, r_mas->offset + 1)) {
                r_mas->last = mas_safe_pivot(r_mas, r_wr_mas->pivots,
                                             r_wr_mas->type, r_mas->offset + 1);
                r_mas->offset++;                        / search j k nav Enter open q back
                r_wr_mas->r_max = r_mas->last;
```

```
        }

Since the goal is to set the end pivot correctly for the next patch set,
should end_piv and offset_end also be updated if r_max is expanded here?
```

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 387,044 · Cached: 2,136,186 · Out: 4,534

View Raw Log ↑ Back to Summary

Patch 10: maple_tree: introduce maple_copy node and use it in mas_spanning_rebalance()

Result: No regressions # Reviewed

▶ Inline Review

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 962,146 · Cached: 4,715,524 · Out: 4,145

View Raw Log ↑ Back to Summary

Patch 11: maple_tree: testing update for spanning store

Result: Critical: 0 · High: 0 · Medium: 2 · Low: 2 # Reviewed

▼ Inline Review

Copy

```
commit 491f8f2c8b2b3f77eb82b44734057839a12cdadf
Author: Liam R. Howlett <Liam.Howlett@oracle.com>

maple_tree: testing update for spanning store

This commit adds explicit tests for spanning store corner cases found during
RCU stress testing. It also adds a dump mutex to prevent console flooding
when multiple threads encounter errors.

Link: https://lkml.kernel.org/r/20260130205935.2559335-13-Liam.Howlett@oracle.com
```

```
> At the same time add some locking for easier visibility of the rcu stress
> testing.  Only a single dump of the tree will happen on the first detected
> issue instead of flooding the console with output.
```

```
Is the dump mutex fully preventing console flooding here?

Looking at rcu_reader_fwd(), the new checks for mas.index and mas.last use
pthread_mutex_trylock(), but the existing printk and RCU_MT_BUG_ON calls
below them in the same loop do not.

If multiple threads hit those existing checks simultaneously, won't they
still bypass the mutex and flood the console?
```

```
> diff --git a/tools/testing/radix-tree/maple.c b/tools/testing/radix-tree/maple.c
> --- a/tools/testing/radix-tree/maple.c
> +++ b/tools/testing/radix-tree/maple.c
[ ... ]
> @@ -33997,8 +33998,25 @@ static void *rcu_reader_fwd(void *ptr)
>                                 }
>                         }
```

/ search j k nav Enter open q back

```
> 
> -                       RCU_MT_BUG_ON(test, mas.index != r_start);
> -                       RCU_MT_BUG_ON(test, mas.last != r_end);
> +                       if (mas.index != r_start) {
> +                               if (pthread_mutex_trylock(&test->dump) != 0) {
> +                                       rcu_read_unlock();
> +                                       goto quit;
> +                               }
> +                               printk("start is wrong: %lx (%lu) vs expected %lx (%lu)\n",
> +                                      mas.index, mas.index, r_start, r_start);
> +                               RCU_MT_BUG_ON(test, mas.index != r_start);
> +                       }
> +
> +                       if (mas.last != r_end) {
> +                               if (pthread_mutex_trylock(&test->dump) != 0) {
```

If a thread encounters both mas.index != r_start and mas.last != r_end in
the same iteration, could this result in undefined behavior?

The test->dump mutex is initialized as PTHREAD_MUTEX_DEFAULT. Since the
mutex is never unlocked in the first block, won't the second
pthread_mutex_trylock() return EBUSY and force a premature thread abort via
the goto quit path instead of reporting the second error?

Additionally, since the mutex is never unlocked, and the vals struct is
shared across multiple test suites like run_check_rcu() and
run_check_rcu_slowread(), won't a failure in the first test permanently lock
the mutex?

Could this cause subsequent tests encountering errors to fail the trylock,
completely skip their validation, and silently hide subsequent bugs?

[...]

```
> +     mtree_lock(mt);
> +     /* Store a null across a boundary that ends in a null */
> +     mas_set(&mas, i); /* 0xC2AB */
> +     MT_BUG_ON(mt, mas_walk(&mas) == NULL);
> +     MT_BUG_ON(mt, mas.end != mas.offset);
> +     MT_BUG_ON(mt, mas_next_range(&mas, ULONG_MAX) != NULL);
> +     mas_set_range(&mas, i, mas.last - 1);
> +     mas_store_gfp(&mas, NULL, GFP_KERNEL);
```

Is it safe to use GFP_KERNEL while holding a spinlock?

The code acquires a spinlock via mtree_lock(mt) just above, and then calls
mas_store_gfp() with GFP_KERNEL. While this is currently in userspace testing
where spinlocks do not disable preemption, using GFP_KERNEL inside a spinlock
violates atomic context rules and would trigger a might_sleep() panic in the
kernel.

This pattern appears to be copied from elsewhere in the test file, but
several new instances are added in this patch. Would it be better to use
GFP_ATOMIC here to align with kernel context rules?

[...]

```
> +     /* Store a null across a boundary that starts and ends next to nulls */
> +     mas_set(&mas, 49800);
> +     MT_BUG_ON(mt, mas_walk(&mas) == NULL);          / search  j  k  nav  Enter  open  q  back
> +     MT_BUG_ON(mt, mas.index != 49800);
```

```
> +     mas_set(&mas, 49815);
> +     MT_BUG_ON(mt, mas_walk(&mas) == NULL);
> +     MT_BUG_ON(mt, mas.last != 49815);
> +     mas_set_range(&mas, 49800, 49815);
> +     mas_store_gfp(&mas, NULL, GFP_KERNEL);
> +     /* Results in 49846-49879: (nil) */
```

> This isn't a bug, but is this comment accurate for this test case?
>
> The comment states the result is in 49846-49879, but the bounds being checked just below are 49796 and 49819. It looks like the comment might have been copied from the preceding test case.

```
> +     MT_BUG_ON(mt, mas.index != 49796);
> +     MT_BUG_ON(mt, mas.last != 49819);
> +     mt_validate(mt);
```

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 69,215 · Cached: 269,169 · Out: 4,663

View Raw Log ↑ Back to Summary

Patch 12: maple_tree: inline mas_spanning_rebalance_loop() into mas_wr_spanning_rebalance()

Result: **No regressions** **# Reviewed**

▶ **Inline Review**

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 343,783 · Cached: 2,648,368 · Out: 2,448

View Raw Log ↑ Back to Summary

Patch 13: maple_tree: change initial big node setup in mas_wr_spanning_rebalance()

Result: **No regressions** **# Reviewed**

▶ **Inline Review**

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 123,152 · Cached: 678,444 · Out: 2,776

View Raw Log ↑ Back to Summary

Patch 14: maple_tree: introduce ma_leaf_max_gap()

Result: **No regressions** **# Reviewed**

▶ **Inline Review**

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 126,479 · Cached: 1,121,191 · Out: 1,273

View Raw Log ↑ Back to Summary

Patch 15: maple_tree: add gap support, slot and pivot sizes for maple copy

/ search j k nav Enter open q back

Result: No regressions # Reviewed

▶ Inline Review

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 1,026,203 · Cached: 6,491,453 · Out: 7,411

View Raw Log ↑ Back to Summary

Patch 16: maple_tree: start using maple copy node for destination

Result: Critical: 1 · High: 2 · Medium: 0 · Low: 0 # Reviewed

▼ Inline Review

Copy

```
commit 4757b90a6dda49dde2bf96027bae008f4ca2a51e
Author: Liam R. Howlett <Liam.Howlett@oracle.com>

maple_tree: start using maple copy node for destination

Stop using the maple subtree state and big node in favour of using three
destinations in the maple copy node. Extract out the sibling init into the
data calculation since this is where the insufficient data can be detected.
Introduce the dst_setup() function which will decide how many nodes are needed.

Link: https://lkml.kernel.org/r/20260130205935.2559335-18-Liam.Howlett@oracle.com

> diff --git a/lib/maple_tree.c b/lib/maple_tree.c
> index 9c701ee7412c..4d9e7f00f5c8 100644
> --- a/lib/maple_tree.c
> +++ b/lib/maple_tree.c

[ ... ]

> +static inline
> +void node_finalise(struct maple_node *node, enum maple_type mt,
> +                   unsigned char end)
> +{
> +     unsigned char max_end = mt_slots[mt];
> +     unsigned char size;
> +     unsigned long *gaps;
> +     unsigned char gap_slot;
> +
> +     gaps = ma_gaps(node, mt);
> +     if (end < max_end - 1) {

If exactly one slot is unused (for example, end is 9 and max_end is 10),
this condition evaluates to false. Does this skip the zeroing operations
and leave the final slot and gap uninitialized? This might cause future
operations to use garbage data from the slab cache.

> +             size = max_end - end;
> +             memset(ma_slots(node, mt) + end, 0, size * sizeof(void *));
> +
> +             if (gaps)
> +                     memset(gaps + end, 0, size * sizeof(unsigned long));
> +
> +             if (--size)
```

```
> +                       memset(ma_pivots(node, mt) + end, 0, size * sizeof(unsigned long));
> +       }
> +
> +       gap_slot = 0;
> +       if (gaps && !ma_is_leaf(mt)) {
> +               unsigned long max_gap;
> +
> +               max_gap = 0;
> +               for (int i = 0; i <= end; i++)
> +                       if (gaps[i] > max_gap) {
```

> Since end represents the count of populated slots, if the node is completely
> full, does this loop iterate one time too many?
>
> It appears this could read past the array bounds of the gaps array and into
> the adjacent meta struct. Could this out-of-bounds index later cause the
> kernel to alias the gap array with slots and use an integer gap size as a
> pointer?

```
> +                               gap_slot = i;
> +                               max_gap = gaps[i];
> +                       }
> +       }
```

> [...]

```
> +static inline
> +void cp_data_write(struct maple_copy *cp, struct ma_state *mas)
> +{
```

> [...]

```
> +                       if (s_offset > src_end) {
> +                               /* This source is exhausted */
> +                               s++;
> +                               if (s >= cp->s_count) {
> +                                       cp->dst[d].max = d_max;
> +                                       node_finalise(dst, d_mt, dst_offset);
> +                                       return;
> +                               }
```

> When all sources are consumed and the function returns early here, does it
> entirely bypass the trailing NULL entry removal located at the bottom of the
> outer loop?
>
> If a spanning store ends with a NULL value across a node boundary, it looks
> like the final destination node might incorrectly retain this trailing NULL.
> Could this leave the tree in an invalid state and trigger an MT_BUG_ON() in
> mt_validate() by violating the rule against NULLs at the end of leaves?

```
> +                               /* Reset local src */
> +                               src = cp->src[s].node;
> +                               s_offset = cp->src[s].start;
> +                               src_end = cp->src[s].end;
> +                               s_max = cp->src[s].max;
> +                               s_mt = cp->src[s].mt;
> +                       }
```

/ search j k nav Enter open q back

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 501,608 · Cached: 4,849,728 · Out: 6,956

View Raw Log ↑ Back to Summary

Patch 17: maple_tree: inline mas_wr_spanning_rebalance()

Result: No regressions # Reviewed

▶ **Inline Review**

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 164,898 · Cached: 722,673 · Out: 845

View Raw Log ↑ Back to Summary

Patch 18: maple_tree: remove unnecessary return statements

Result: No regressions # Reviewed

▶ **Inline Review**

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 42,351 · Cached: 130,211 · Out: 138

View Raw Log ↑ Back to Summary

Patch 19: maple_tree: separate wr_split_store and wr_rebalance store type code path

Result: Critical: 0 · High: 0 · Medium: 1 · Low: 0 # Reviewed

▼ **Inline Review**

Copy

```
commit 366334c7bfbceec3dbdec894c86cbacc92004279
Author: Liam R. Howlett <Liam.Howlett@oracle.com>

maple_tree: separate wr_split_store and wr_rebalance store type code path

The split and rebalance store types both go through the same function that
uses the big node. Separate the code paths so that each can be updated
independently.

Link: https://lkml.kernel.org/r/20260130205935.2559335-21-Liam.Howlett@oracle.com

> diff --git a/lib/maple_tree.c b/lib/maple_tree.c
> index 87bbc5492305..2383c97f684b 100644
> --- a/lib/maple_tree.c
> +++ b/lib/maple_tree.c

[ ... ]

> +/*
> + * mas_wr_rebalance() - Insufficient data in one node needs to either get data
> + * from a sibling or absorb a sibling all together.
> + * @wr_mas: The write maple state
> + */
> +static noinline_for_kasan void mas_wr_rebalance(struct ma_wr_state *wr_mas)
> +{
> +     struct maple_big_node b_node;
```

/ search j k nav Enter open q back

```
> +
> +     trace_ma_write(__func__, wr_mas->mas, 0, wr_mas->entry);
```

> Does this break userspace tracing output resolution?
>
> The trace_ma_write tracepoint uses a pointer to store the function name
> rather than capturing the string contents in the ring buffer.
>
> For offline tracing tools like perf script or trace-cmd to resolve this
> pointer, the string must be registered in the __tracepoint_str section. This
> is guaranteed by using the TP_FCT macro instead of __func__. Bypassing
> tracepoint_string() by passing __func__ directly causes trace parsers to fail
> to resolve the function name, resulting in a fault or raw hex addresses in
> the tracing output.
>
> I noticed this was later fixed in the series by commit 2498fa5328c9
> (maple_tree: use maple copy node for mas_wr_rebalance() operation), but
> wanted to point it out in case it causes issues during bisection.

```
> +     memset(&b_node, 0, sizeof(struct maple_big_node));
> +     mas_store_b_node(wr_mas, &b_node, wr_mas->offset_end);
> +     WARN_ON_ONCE(wr_mas->mas->store_type != wr_rebalance);
> +     return mas_rebalance(wr_mas->mas, &b_node);
>  }
```

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 374,089 · Cached: 2,660,493 · Out: 4,714

View Raw Log ↑ Back to Summary

Patch 20: maple_tree: add cp_is_new_root() helper

Result: No regressions **# Reviewed**

▶ **Inline Review**

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 90,009 · Cached: 479,898 · Out: 683

View Raw Log ↑ Back to Summary

Patch 21: maple_tree: use maple copy node for mas_wr_rebalance() operation

Result: Critical: 1 · High: 0 · Medium: 0 · Low: 0 **# Reviewed**

▼ **Inline Review**

Copy

> commit 2498fa5328c94efb95caa7b6fd5d0b123b7797f1
> Author: Liam R. Howlett <Liam.Howlett@oracle.com>
>
> maple_tree: use maple copy node for mas_wr_rebalance() operation
>
> This commit updates the rebalance operations to use the maple copy
> node instead of the big node. It aligns the code more closely with
> spanning stores by using new helper functions for data calculation
> and ascending the tree.
>
> Link: https://lkml.kernel.org/r/20260130205935.255933[illegible]5-23-Liam.Howlett@oracle.com

/ search j k nav Enter open q back

```
> diff --git a/lib/maple_tree.c b/lib/maple_tree.c
> index 6a8f75dac6c9..7ce7c0a33943 100644
> --- a/lib/maple_tree.c
> +++ b/lib/maple_tree.c
```

> [...]

```
> @@ -4379,16 +4547,47 @@ static noinline_for_kasan void mas_wr_split(struct ma_wr_state
*wr_mas)
>   * mas_wr_rebalance() - Insufficient data in one node needs to either get data
>   * from a sibling or absorb a sibling all together.
>   * @wr_mas: The write maple state
> + *
> + * Rebalance is different than a spanning store in that the write state is
> + * already at the leaf node that's being altered.
>   */
> -static noinline_for_kasan void mas_wr_rebalance(struct ma_wr_state *wr_mas)
> +static void mas_wr_rebalance(struct ma_wr_state *wr_mas)
>  {
> -     struct maple_big_node b_node;
> -
> -     trace_ma_write(__func__, wr_mas->mas, 0, wr_mas->entry);
> -     memset(&b_node, 0, sizeof(struct maple_big_node));
> -     mas_store_b_node(wr_mas, &b_node, wr_mas->offset_end);
> -     WARN_ON_ONCE(wr_mas->mas->store_type != wr_rebalance);
> -     return mas_rebalance(wr_mas->mas, &b_node);
> +     struct maple_enode *old_enode;
> +     struct ma_state parent;
> +     struct ma_state *mas;
> +     struct maple_copy cp;
> +     struct ma_state sib;
>
> -     trace_ma_write(__func__, wr_mas->mas, 0, wr_mas->entry);
> -     memset(&b_node, 0, sizeof(struct maple_big_node));
> -     mas_store_b_node(wr_mas, &b_node, wr_mas->offset_end);
> -     WARN_ON_ONCE(wr_mas->mas->store_type != wr_rebalance);
> -     return mas_rebalance(wr_mas->mas, &b_node);
> +     /*
> +      * Rebalancing occurs if a node is insufficient.  Data is rebalanced
> +      * against the node to the right if it exists, otherwise the node to the
> +      * left of this node is rebalanced against this node.  If rebalancing
> +      * causes just one node to be produced instead of two, then the parent
> +      * is also examined and rebalanced if it is insufficient.  Every level
> +      * tries to combine the data in the same way.  If one node contains the
> +      * entire range of the tree, then that node is used as a new root node.
> +      */
> +
> +     mas = wr_mas->mas;
> +     trace_ma_op(TP_FCT, mas);
> +     parent = *mas;
> +     cp_leaf_init(&cp, mas, wr_mas, wr_mas);
> +     do {
> +             if (!mte_is_root(parent.node)) {
> +                     mas_ascend(&parent);
> +                     parent.end = mas_data_end(&parent);
> +             }
> +             rebalance_data(&cp, wr_mas, &sib, &parent);
> +             multi_src_setup(&cp, wr_mas, wr_mas, &sib);
```

> Could this lead to data corruption and truncation of valid elements during
> bottom-up tree rebalancing?
>
> When ascending the tree after merging a node with its sibling, wr_mas is
> passed as both the left and right source states to multi_src_setup().
>
> In rebalance_ascend(), mas->offset is set to the left-most replaced slot
> (parent->offset or parent->offset - 1), while wr_mas->offset_end is set
> to the right-most replaced slot.
>
> While cp_data_calc() correctly calculates the target size by skipping the
> fully replaced range (using r_wr_mas->offset_end), multi_src_setup()
> incorrectly copies the right side starting from r_wr_mas->mas->offset + 1
> (which evaluates to mas->offset + 1 since r_wr_mas is wr_mas).
>
> This results in copying from the right sibling or original node that was
> already merged into the new children, rather than starting from
> offset_end + 1. Consequently, multi_src_setup() appends the already-merged
> node to the source list, providing more elements than cp->data allocated
> space for.
>
> Does this cause cp_data_write() to truncate the copy when it hits cp->data,
> silently dropping the final valid elements at the end of the old node?

```
> +             dst_setup(&cp, mas, wr_mas->type);
> +             cp_data_write(&cp, mas);
> +     } while (rebalance_ascend(&cp, wr_mas, &sib, &parent));
> +
> +     old_enode = mas->node;
> +     mas->node = mt_slot_locked(mas->tree, cp.slot, 0);
> +     mas_wmb_replace(mas, old_enode, cp.height);
> +     mtree_range_walk(mas);
>  }
```

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 1,604,858 · Cached: 8,684,369 · Out: 6,148

View Raw Log ↑ Back to Summary

Patch 22: maple_tree: add test for rebalance calculation off-by-one

Result: Critical: 0 · High: 0 · Medium: 1 · Low: 0 **# Reviewed**

▼ Inline Review

Copy

> commit 37ea8e0708ff6e5e1cdf08077bac7bd7abbb409c
> Author: Liam R. Howlett <Liam.Howlett@oracle.com>
>
> maple_tree: add test for rebalance calculation off-by-one
>
> This commit adds a test to recreate a scenario where an incorrect rebalance
> step went too far up the tree causing insufficient nodes.
>
> Link: https://lkml.kernel.org/r/20260130205935.2559335-24-Liam.Howlett@oracle.com

> diff --git a/tools/testing/radix-tree/maple.c b/tools/testing/radix-tree/maple.c
> index dfd7099f0d8e..5ea45d67556a 100644 / search j k nav Enter open q back
> --- a/tools/testing/radix-tree/maple.c

> +++ b/tools/testing/radix-tree/maple.c

[...]

> @@ -35888,6 +35888,127 @@ static __init int build_full_tree(struct maple_tree *mt, unsigned int flags,
> +static noinline void __init check_erase_rebalance(struct maple_tree *mt)
> +{

[...]

> +	/* Move to Node E */
> +	mas_ascend(&mas);
> +	val = mas.max;
> +	MAS_BUG_ON(&mas, mas_data_end(&mas) > 9);

Will this regression crash the test suite on 32-bit architectures?

On 32-bit systems, MAPLE_ARANGE64_SLOTS is defined as 21. When build_full_tree() initially populates the tree, internal nodes will contain up to 21 entries. When the test ascends to Node E, mas_data_end(&mas) could evaluate to greater than 9 (for example, 20), which would immediately trigger this BUG_ON().

Additionally, erasing node elements down to 4 drops the node far below the 32-bit minimum capacity requirement. Could this trigger cascading, unintended rebalancing that alters the hand-crafted topological assumptions of the test?

[...]

> @@ -36249,6 +36370,10 @@ void farmer_tests(void)
> 	check_mtree_dup(&tree);
> 	mtree_destroy(&tree);
>
> +	mt_init_flags(&tree, MT_FLAGS_ALLOC_RANGE);
> +	check_erase_rebalance(&tree);
> +	mtree_destroy(&tree);
> +
> 	/* RCU testing */
> 	mt_init_flags(&tree, 0);

Should this test call be guarded by an if (!MAPLE_32BIT) check to prevent the hardcoded 64-bit capacity assumptions from causing unintended crashes on 32-bit systems?

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 540,494 · Cached: 3,108,785 · Out: 4,015

View Raw Log ↑ Back to Summary

Patch 23: maple_tree: add copy_tree_location() helper

Result: No regressions **# Reviewed**

▶ **Inline Review**

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 42,674 · Cached: 97,511 · Out: 129

/ search j k nav Enter open q back

View Raw Log ↑ Back to Summary

Patch 24: maple_tree: add cp_converged() helper

Result: No regressions # Reviewed

▶ Inline Review

To: None Cc: None Status: Disabled (SMTP not configured)
Tokens used: In: 63,463 · Cached: 121,999 · Out: 150

View Raw Log ↑ Back to Summary

Patch 25: maple_tree: use maple copy node for mas_wr_split()

Result: No regressions # Reviewed

▶ Inline Review

To: None Cc: None Status: Disabled (SMTP not configured)
Tokens used: In: 2,004,111 · Cached: 12,893,576 · Out: 5,211

View Raw Log ↑ Back to Summary

Patch 26: maple_tree: remove maple big node and subtree structs

Result: No regressions # Reviewed

▶ Inline Review

To: None Cc: None Status: Disabled (SMTP not configured)
Tokens used: In: 157,579 · Cached: 1,408,445 · Out: 1,212

View Raw Log ↑ Back to Summary

Patch 27: maple_tree: pass maple copy node to mas_wmb_replace()

Result: No regressions # Reviewed

▶ Inline Review

To: None Cc: None Status: Disabled (SMTP not configured)
Tokens used: In: 381,995 · Cached: 3,100,039 · Out: 3,146

View Raw Log ↑ Back to Summary

Patch 28: maple_tree: don't pass end to mas_wr_append()

Result: No regressions # Reviewed

▶ Inline Review

To: None Cc: None Status: Disabled (SMTP not configured)
Tokens used: In: 67,178 · Cached: 203,641 · Out: 455

View Raw Log ↑ Back to Summary

/ search j k nav Enter open q back

Patch 29: maple_tree: clean up mas_wr_node_store()

Result: No regressions # Reviewed

▶ Inline Review

To: None **Cc:** None **Status:** Disabled (SMTP not configured)
Tokens used: In: 409,311 · Cached: 4,661,108 · Out: 87,343

View Raw Log ↑ Back to Summary