Why is arrow utf8_neq kernel slower than native Rust

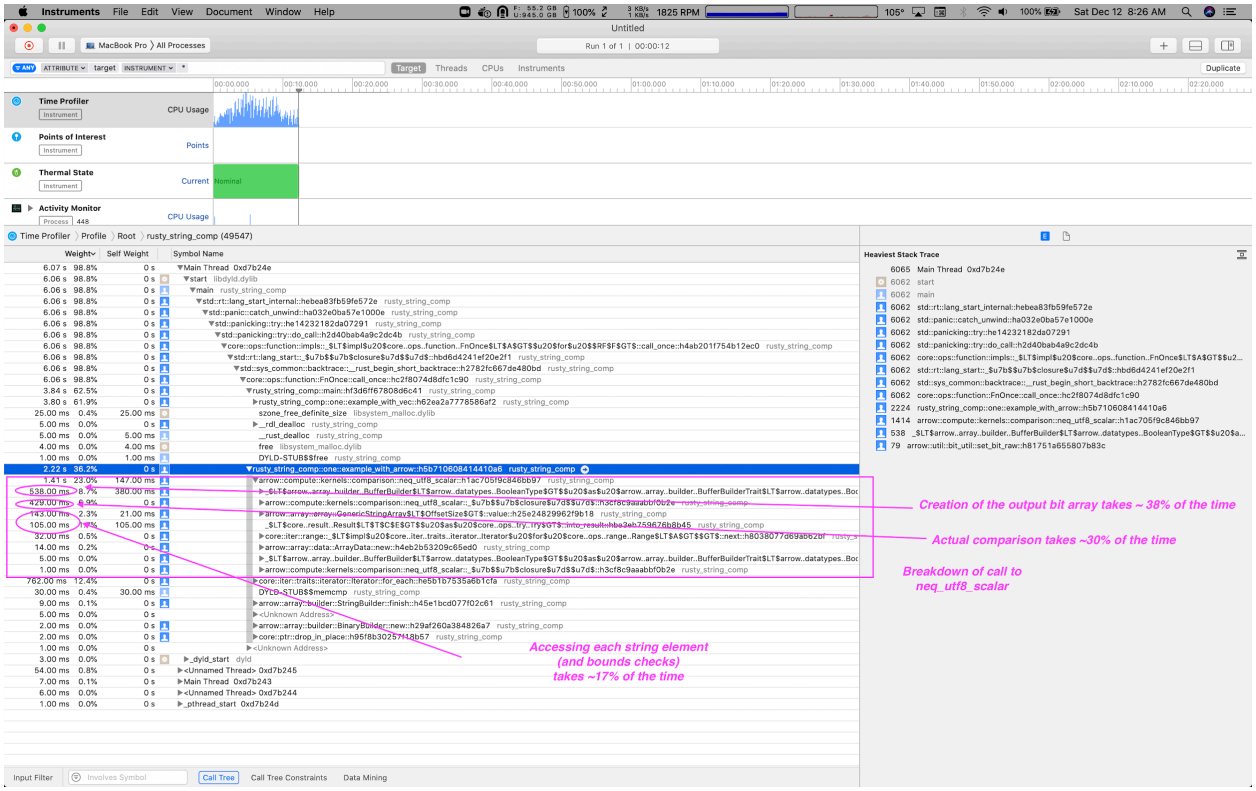
Dec 12, 2020
Andrew Lamb

[Update (Dec 13): There is a PR up for arrow that brings the performance closer to parity -- see Dec 13, 2020 update section]

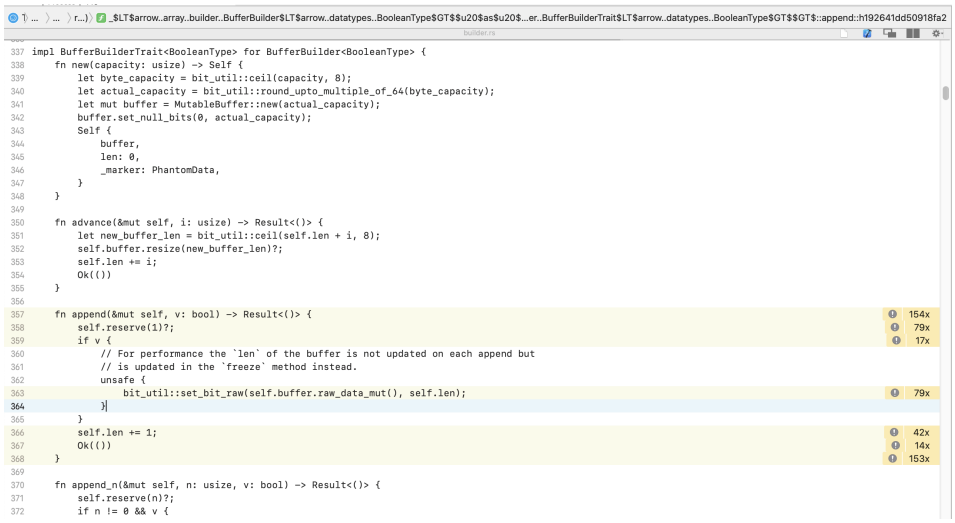
In the Rusty intro to Arrow InfluxDB IOx Tech Talks – December 2020 edition, and (slides) we observed that arrow is slower than native Rust for string comparison and I hand waved it away as it was a minor point in a larger discussion.

rdettai@gmail.com brought it up in an email, and we looked into the discrepancy more.

It turns out that most of the difference can be accounted for by the time needed to create the output arrow array. Running under Instruments on my laptop, fully 38 percent of the time is actually spent creating the output bitset:



When I dug into it a little, there was a bunch of seemingly low hanging fruit. For example ensuring the size was reserved once up front rather than each time.



```
impl BufferBuilderTrait<BooleanType> for BufferBuilder<BooleanType> {
    fn new(capacity: usize) -> Self {
        let byte_capacity = bit_util::ceil(capacity, 8);
        let actual_capacity = bit_util::round_upto_multiple_of_64(byte_capacity);
        let mut buffer = MutableBuffer::new(actual_capacity);
        buffer.set_null_bits(0, actual_capacity);
        Self {
            buffer,
            len: 0,
            _marker: PhantomData,
        }
    }

    fn advance(&mut self, i: usize) -> Result<()> {
        let new_buffer_len = bit_util::ceil(self.len + i, 8);
        self.buffer.resize(new_buffer_len)?;
        self.len += i;
        Ok(())
    }

    fn append(&mut self, v: bool) -> Result<()> {                                  154x
        self.reserve(1)?;                                                          79x
        if v {                                                                     17x
            // For performance the `len` of the buffer is not updated on each append but
            // is updated in the `freeze` method instead.
            unsafe {
                bit_util::set_bit_raw(self.buffer.raw_data_mut(), self.len);       79x
            }
        }
        self.len += 1;                                                             42x
        Ok(())                                                                     14x
    }                                                                              153x

    fn append_n(&mut self, n: usize, v: bool) -> Result<()> {
        self.reserve(n)?;
        if n != 0 && v {
```

Experimental setup:

The code I used can be found here: https://github.com/alamb/arrow_string_comp and did two things:

1. Profile with instruments to see where the time (in arrow) is going
2. Run against latest arrow mater (rather than 2.0 release)

The program makes 20M instances of 3 distinct strings and then filters out 1 of 3 of those 10 times in a loop

Here is what **example_with_vec** does:

```
let not_west_bitset: Vec<bool> = string_vec
    .iter()
    .map(|s| s != "us-west")
    .collect();
```

Here is what **example_with_arrow** does:

```
let not_west_bitset = neq_utf8_scalar(&array, "us-west").unwrap();
```

Here is the output:

```
cargo run --release

Hello, world!
example_with_vec
created array with 20000000 elements in 1.096240239s
Completed finding bitset: 20000000 elements in 62.169064ms
Completed finding bitset: 20000000 elements in 51.469797ms
Completed finding bitset: 20000000 elements in 51.104856ms
Completed finding bitset: 20000000 elements in 53.626129ms
Completed finding bitset: 20000000 elements in 55.967121ms
Completed finding bitset: 20000000 elements in 56.630698ms
Completed finding bitset: 20000000 elements in 57.243619ms
Completed finding bitset: 20000000 elements in 55.918409ms
Completed finding bitset: 20000000 elements in 57.317761ms
Completed finding bitset: 20000000 elements in 58.78041ms

example_with_arrow
created array with 20000000 elements in 688.184769ms
Found 20000000 not in west in 141.382892ms
Found 20000000 not in west in 139.243362ms
Found 20000000 not in west in 135.835819ms
Found 20000000 not in west in 134.969208ms
Found 20000000 not in west in 136.255583ms
Found 20000000 not in west in 134.158282ms
Found 20000000 not in west in 136.794011ms
Found 20000000 not in west in 134.795349ms
Found 20000000 not in west in 137.669832ms
Found 20000000 not in west in 133.383716ms
```

I also tried the same experiment on arrow master at db20c7a611adac7be5cdd9350792852345f5b6b4 and it turns out the performance has actually slowed down a bit.

```
example_with_arrow
created array with 20000000 elements in 1.346098327s
Found 20000000 not in west in 180.012658ms
Found 20000000 not in west in 175.848718ms
Found 20000000 not in west in 178.413715ms
Found 20000000 not in west in 173.68871ms
Found 20000000 not in west in 179.408338ms
Found 20000000 not in west in 176.348492ms
Found 20000000 not in west in 173.9123ms
Found 20000000 not in west in 176.258987ms
```

```
Found 20000000 not in west in 173.501113ms
Found 20000000 not in west in 172.746955ms
```

Update December 13, 2020: community is working!

rdettai@gmail.com points at this PR from Daniël Heres https://github.com/apache/arrow/pull/8900 which speeds things up (largely by avoiding the Builder).

And when using the code in commit/23c8ff28e56ccb381bcbb321dcbbb946d8fd7db0, the output now shows arrow almost at par with basic rust (and still handling nulls, etc):

```
Hello, world!
example_with_vec
created array with 20000000 elements in 1.200794699s
Completed finding bitset: 20000000 elements in 82.056189ms
Completed finding bitset: 20000000 elements in 59.892746ms
Completed finding bitset: 20000000 elements in 53.485284ms
Completed finding bitset: 20000000 elements in 62.333645ms
Completed finding bitset: 20000000 elements in 55.681219ms
Completed finding bitset: 20000000 elements in 55.758439ms
Completed finding bitset: 20000000 elements in 53.423546ms
Completed finding bitset: 20000000 elements in 53.974439ms
Completed finding bitset: 20000000 elements in 53.547294ms
Completed finding bitset: 20000000 elements in 55.003746ms
example_with_arrow
created array with 20000000 elements in 801.781593ms
Found 20000000 not in west in 85.545518ms
Found 20000000 not in west in 77.341209ms
Found 20000000 not in west in 80.133018ms
Found 20000000 not in west in 80.703599ms
Found 20000000 not in west in 81.245902ms
Found 20000000 not in west in 79.48747ms
Found 20000000 not in west in 78.57811ms
Found 20000000 not in west in 78.758314ms
Found 20000000 not in west in 77.196055ms
Found 20000000 not in west in 78.034662ms
```