GRAPHITE-RUST

Xavier Lange

xrlange@tureus.com

xavier.lange@viasat.com

@tureus

ME

- Independent software contractor
- Computer engineering background
  - Working with ViaSat, Inc. on large scale logging
- Dig the outdoors
- Cat owner

MY FAVORITE LANGUAGES, IN NO PARTICULAR ORDER

- OO
  - Ruby
- Functional
  - Erlang
  - Haskell
- Weird
  - Rust

iangreenleaf 2:47 PM

Come on, how many languages can you possibly juggle? This is ridiculous.

xrl 2:47 PM

watch me 😎

iangreenleaf 2:47 PM

I swear @xrl just picks a Language Of The Week

xrl 2:48 PM

I deliver solutions not code 😎 (edited)

iangreenleaf 2:48 PM

solutions, in whichever language is LOTW this week

MY CAT LUA

Inspiration to stop lounging and go for it

Even if you might get hit by a car

Lua watching standup comedy

GRAPHITE-RUST

- Take a "slow" Python implementation and clone in Rust
- Built to satisfy some needs at ViaSat, Inc.
  - Love graphite
  - Dislike it's opaque performance configuration
  - Troublesome installation
- A dash of NIH
- Supports a private cloud installation and large-ish logging/indexing cluster

TIME SERIES AS A RUST TYPE

Pseudo code

```
fn main() {
    let retention = 60*60*24; // one day
    let mut archive : Vec<(u32,f64)> = Vec::with_capacity(retentio

    let my_series = &mut archive[..];

    my_series[0] = (1438460243,100.0);
}
```

WE'LL COVER

- Graphite concepts (run don't walk)
- Analysis of graphite arch shortcomings (brisk walk)
- The Rust-Graphite implementation (some examples, leisurely strole)
- Analysis of graphite-rust shortcomings (leisurely strole)

LET'S TALK METRICS

NOT

- Analytics
  - Urchin/Google Analytics
- Business Intelligence
  - Hadoop/Storm
- High-level
  - Web session heatmaps
- Aggregation
  - StatsD

GRAPHITE METRIC CONCEPTS

- Long-lived time series
  - Ex: Measure server A's CPU for 2 years
- Regular update intervals
  - Metric is offered every X seconds (our happy-ish spot: 60s)
- Pre-allocated space on disk
- Basic metric naming
  - one-dimensional
  - ad-hoc

NAMES

Metric names are primitive

```
proglangs.rust.community.active_members
proglangs.rust.community.lifetime_questions
proglangs.go.community.active_members
proglangs.go.community.segfault_questions
```

NAMES

Finding your metrics can get fancy

```
proglangs.*.community.active_members
```

Yields

```
-> proglangs.rust.community.active_members
-> proglangs.go.community.active_members
```

VALUES

primitive like whoa

```
(u32,f64)
```

VALUES

u32: seconds since epoch. SECONDS

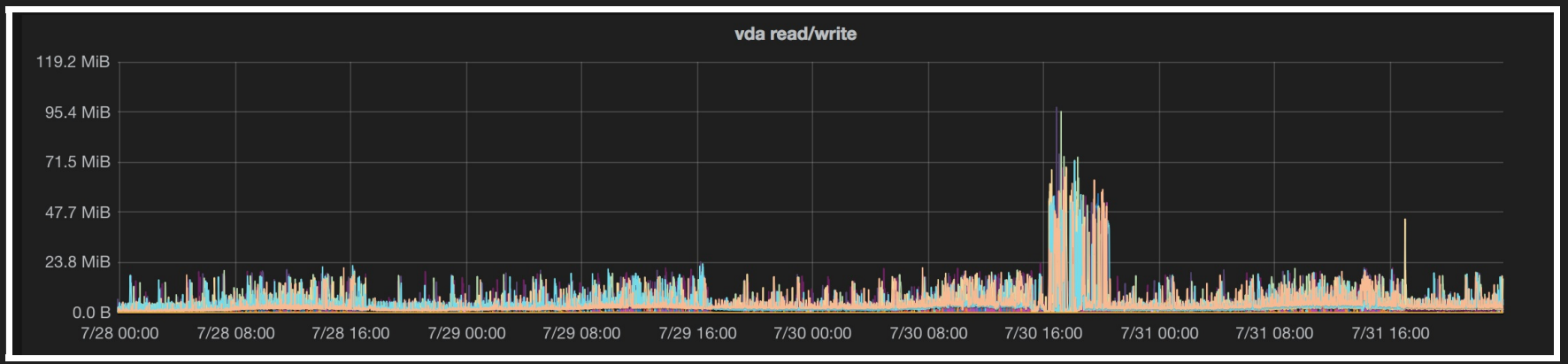
f64: a number, big or small or in between

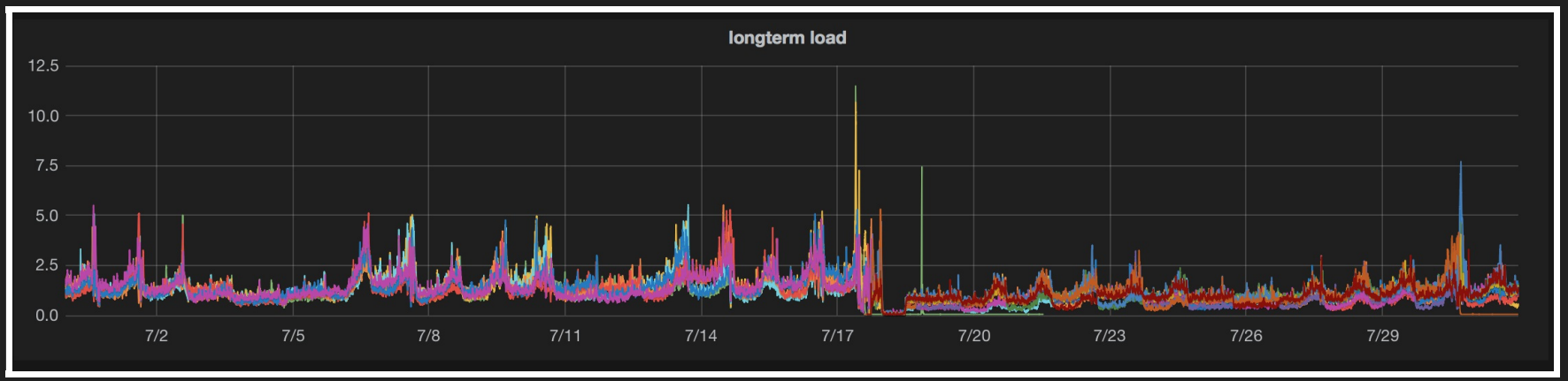
VALUES

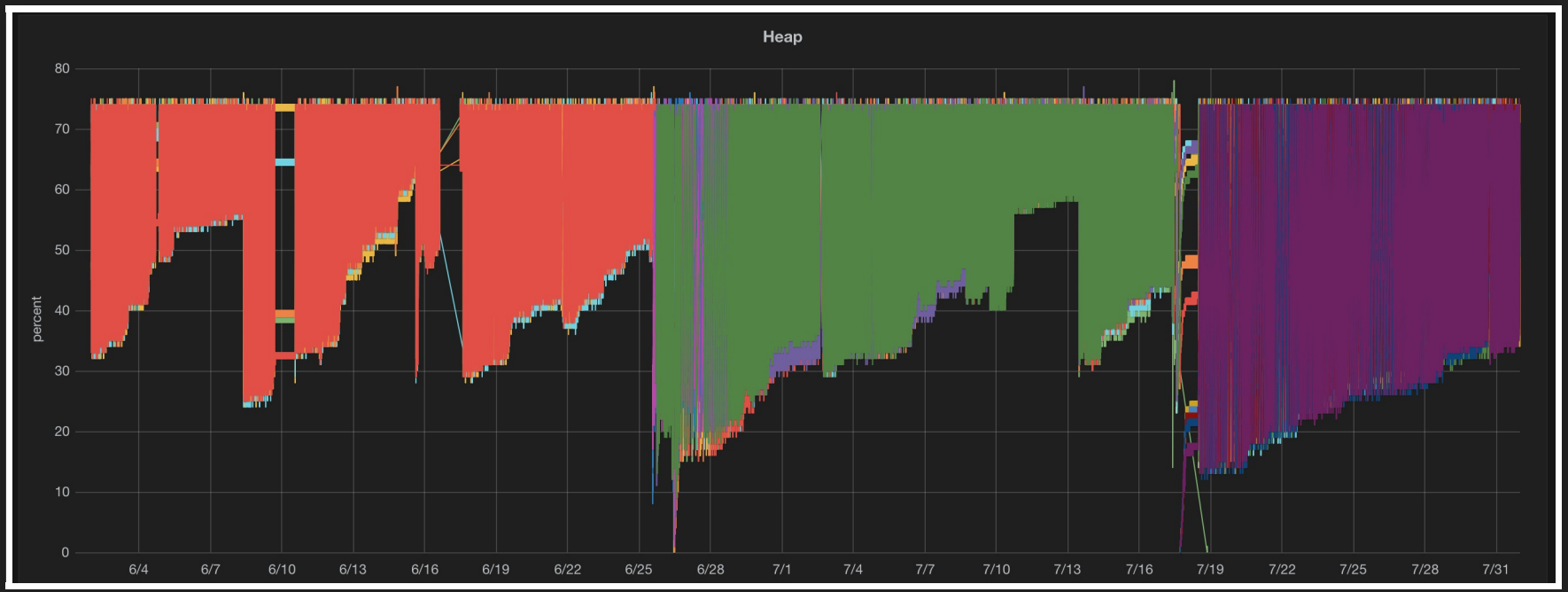
but what can you possibly store with that?

...

enough to be dangerous!


vda read/write
119.2 MiB
95.4 MiB
71.5 MiB
47.7 MiB
23.8 MiB
0.0 B
7/28 00:00
7/28 08:00
7/28 16:00
7/29 00:00
7/29 08:00
7/29 16:00
7/30 00:00
7/30 08:00
7/30 16:00
7/31 00:00
7/31 08:00
7/31 16:00


longterm load
12.5
10.0
7.5
5.0
2.5
0.0
7/2
7/5
7/8
7/11
7/14
7/17
7/20
7/23
7/26
7/29


Heap
percent
80
70
60
50
40
30
20
10
0
6/4
6/7
6/10
6/13
6/16
6/19
6/22
6/25
6/28
7/1
7/4
7/7
7/10
7/13
7/16
7/19
7/22
7/25
7/28
7/31

GRAPHITE IN PYTHON

Actually 3 core components, whose names I have adopted

1. Graphite-Web
2. Carbon
3. Whisper

GRAPHTE IN PYTHON INSTALLATION

Tricky, many components

Run `sudo pip install` or learn the `virtualenv` toolchain

3. GRAPHITE-WEB

A multi-format web service written on top of Django.

- Least interesting to me
- Read-only access to database
- HTML interface for rendering queries
- REST-y interface
  - discovering metrics
  - querying/transform and delivering
  - JSON, CSV, plaintext, pickle, etc
- Tons of features, hidden gems

2. CARBON

A TCP/UDP daemon for recording datapoints

- Moderately interesting to me
- Creates metric storage if necessary
- Writes metrics to disk
- Handles the life-cycle of resources when writing

3. WHISPER

- Fascinating
- Understands bytes on disk
- Writes points
  - Through all archives
- Should be easier than writing a ACID MVCC
  - Like, way easier
  - No real query planner
  - No multi-block joins
  - No transactions

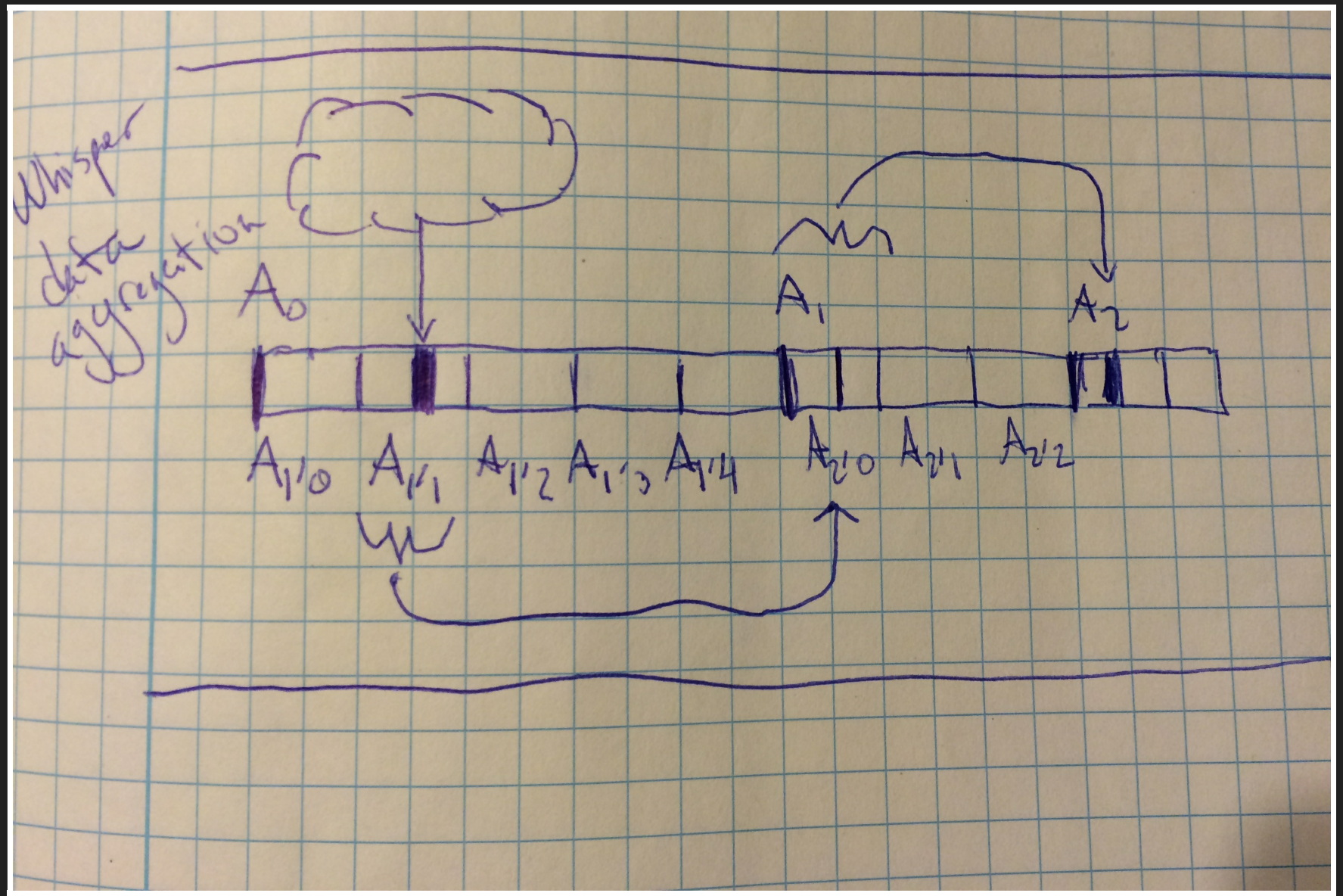
3. WHISPER

... Rust is fast. Maybe that'll make perf automaticly faster?

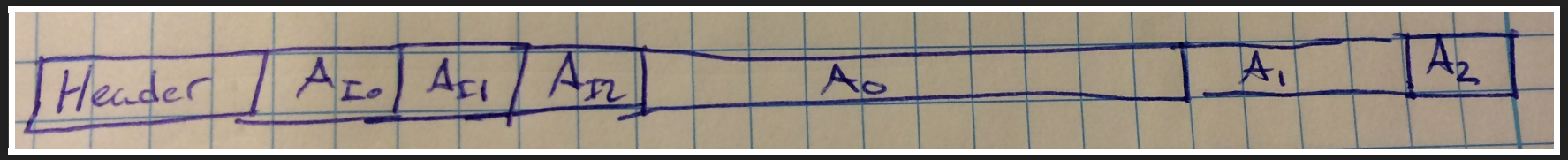
Turned out to be true

DIGGING IN TO WHISPER

VALUE PROPAGATION



INVIDUAL ARCHIVE



```rust
pub trait WhisperFile {
    fn open(&Path) -> Result<Self, Error>;
    fn new(path: &Path, schema: Schema) -> Result<Self, Error>;
    fn write(&mut self, current_time: u64, point: point::Point);
}
```

```rust
pub struct MutexWhisperFile {
    pub handle: Mutex<File>,
    pub header: Header
}
```

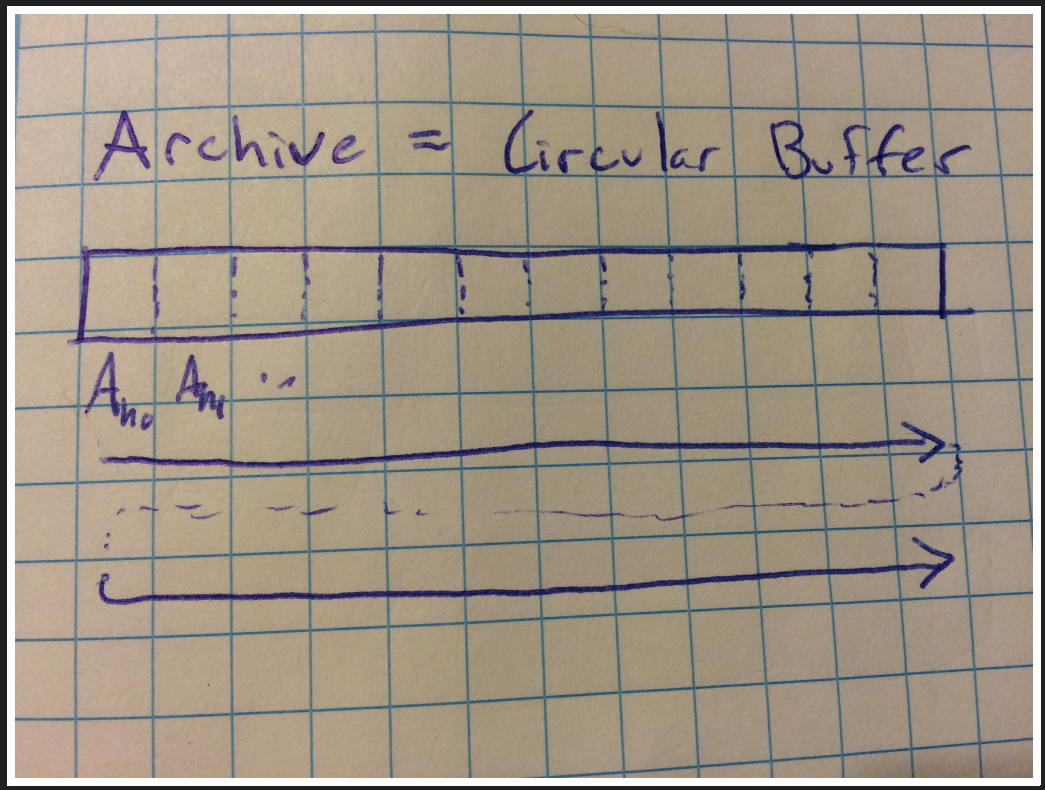
```rust
// TODO: Don't think we need Copy/Clone.
// Just added it to make tests easier to write.
#[derive(PartialEq,Copy,Clone,Debug)]
pub struct ArchiveInfo {
    pub offset: SeekFrom,
    pub seconds_per_point: u64,
    pub points: u64,
    pub retention: u64,
}
```

TUPLE STRUCTS

Whisper constructs not present in Python

```
// Index in to an archive, 0..points.len()
#[derive(Debug, PartialEq, PartialOrd)]
pub struct ArchiveIndex(pub u64);

// A normalized timestamp
pub struct BucketName(pub u64);
```


Archive = Circular Buffer
A_{n_0} A_{n_1} ..

CASE STUDY: WRAP AROUND READ

Used in downsampling

1. Example of "read" amplification
2. I like "pure" code
3. I was figuring out tangled python
4. I wanted to reuse a buffer

CASE STUDY: WRAP AROUND READ FOR DOWNSAMPLING

Type signatures are neat!

```
pub fn read_ops<'a> (h_res_archive: &ArchiveInfo,
                     l_res_archive: &ArchiveInfo,
                     h_res_points: &'a mut [point::Point],
                     point_timestamp: u64,
                     h_res_anchor: BucketName)
-> (
              (ArchiveIndex, &'a mut [point::Point]),
      Option<(ArchiveIndex, &'a mut [point::Point])>
   )
```

CASE STUDY: WRAP AROUND READ

Powered by a favorite: `split_at_mut`

```
// Contiguous read. The easy one.
if h_res_start_index < h_res_end_index {
    ((h_res_start_index, &mut h_res_points[..]), None)
// Wrap-around read
} else {
    let split_index = (hres.points - start_index.0) as usize;
    let (first_buf, second_buf) = points.split_at_mut( split_index
    let zero_index = ArchiveIndex(0);
    ((h_res_start_index, first_buf), Some((zero_index, second_buf)
}
```

IS IT FASTER?

- Yes. Duh.
- 2-3x faster.
- Python: spends more time in userland
- Rust: spends more time in syscalls
- With almost the same naïve behavior as python
- Poor buffer reuse
- Open file, read headers, close it right away

IS IT EASIER TO DISTRIBUTE?

Yes. Duh.

```
$ du -hs target/release/{whisper,carbon,graphite}
1.1M  target/release/whisper
1.2M  target/release/carbon
2.0M  target/release/graphite
```

WHERE DO WE STAND IN FEATURES TODAY?

CARBON

Uses of the WhisperCache

- Hold files descriptors for longer
- Still need eviction
- Make re-use fast
- Does periodic fsync
- Hides creation/reuse

CARBON'S CACHE

```
pub struct Cache {
    pub base_path: PathBuf,
    open_files: HashMap< PathBuf, Box< MutexWhisperFile > >
}
```

GRAPHITE

Not very usable. Having issues with my libraries.

You can still use the Python implementation.

NEXT STEPS

Keep improving write performance

- mmap data, manual
- get better at dtrace
- carry buffers around

WAS RUST A GOOD CHOICE?

I don't like to advocate rewrites. Especially since I didn't write the first impl.

Rust has great tooling, tests, benchmarks.

I can use mmap.

I can reuse buffers.

I have a compiler yell at me.

So I'm happy.

ANYWAY

Rendering a ton of metrics is my current bottleneck in JS land

Server side rendering to PNG could be interesting

THANKS

twitter: @tureus

email: xrlange@gmail.com

graphite-rust: https://github.com/tureus/graphite-rust