Observing when it pays off to do nothing

the insanity-inducing rsync chronicles

rfong@honeycomb.io

Background context

Honeycomb uses a **distributed column-oriented datastore** we call *Retriever*.

Multiple *Retriever* **replicas** syncing/serving, for fault-tolerance.

Why did we build our own datastore? That's a much longer discussion.
tl;dr: we needed a *distributed*, *schemaless*, *fault-tolerant datastore* capable of making *ultra-performant queries over data of arbitrary cardinality and sparseness*.

Varstring compression project

a.k.a. “don’t let our custom distributed datastore crash spectacularly after a minor change to its core record format”

zero-tolerance policy for customer data loss

Architected the changes to be:

1. fully reversible
2. backwards-compatible
3. testable in production

(not pictured here: massive fault-tolerance angst, multiple redesigns, testing in production)

Varstring compression project

Success!

- Overall data footprint 55% smaller
- No noticeable performance hit
- Read/writepath fully reversible and testable in production

Legacy datafile (stream-appended)

Compact-format datafile (atomically written)

Varstring compression project

*The actual **cost-saving** part: need to delete these when safe*

Legacy datafile (stream-appended)

Compact-format datafile (atomically written)

Legacy file deletion

Fault tolerant except for one bootstrap case - multi stage `rsync` to synchronize replicas

4 solid days of p a r a n o i a

- reading `rsync` documentation, `flock` documentation
- pestering entire team for shared knowledge (things didn't work the way we thought they did)
- meticulously reading our replica-syncing code and tests
- writing shell scripts to hackily simulate unmentionable `rsync` failure cases on my local instance

Solution angst

Add distributed fault tolerance to multi stage `rsync`??

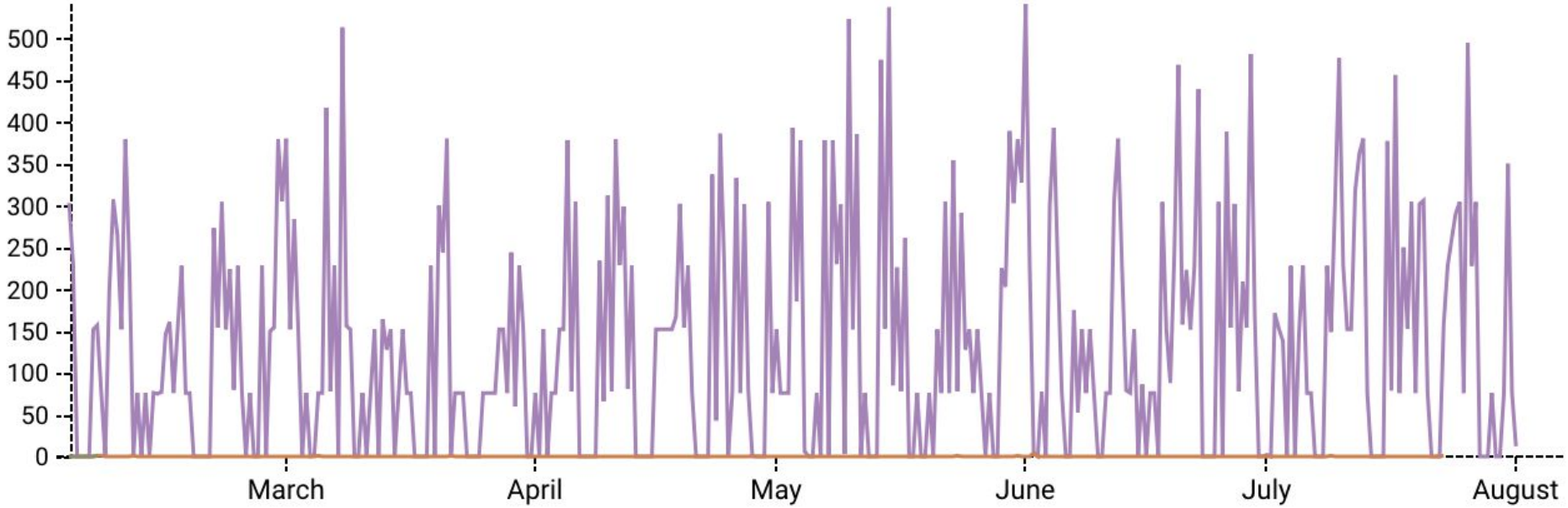
INNUMERABLE CONCERNS

vs.

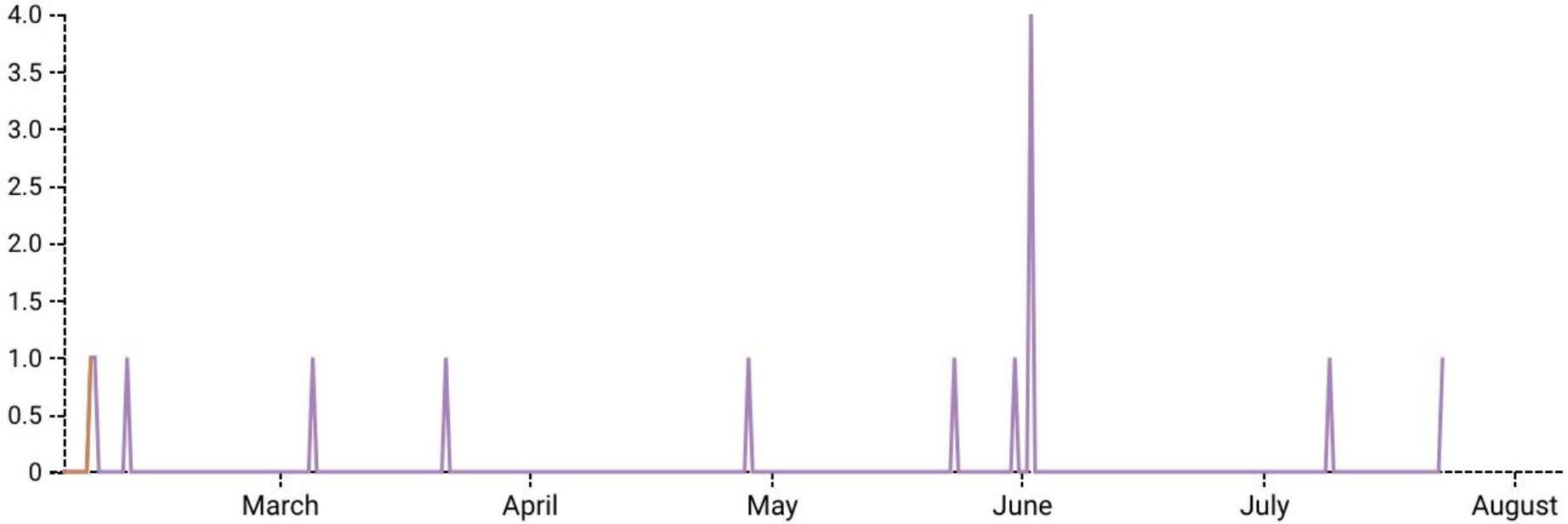
Somehow use `flock` as hacky filesystem concurrency hack?

EVEN WORSE

Retriever bootstrap frequency



Retriever bootstrap with `rsync`



Thank you instrumentation gods

Retriever bootstrap with `rsync`

A full rebuild from `rsync`ing only occurs **2-3x/month**

Represents **< 0.01%** of cumulative uptime across replica groups

Original goal: Improve storage
Compression will improve storage **55%** anyway!

→ Not worth agonizing. Bypass deletion step!

Solution angst

Add distributed fault tolerance to multi stage `rsync`??

INNUMERABLE CONCERNS

vs.

Somehow use `flock` as hacky filesystem concurrency hack?

EVEN WORSE

Must ENGINEER a solution

Solution ~~angst~~ anticlimax

Add distributed fault tolerance to multi stage `rsync`??

INNUMERABLE CONCERNS

vs.

Somehow use `flock` as hacky filesystem concurrency hack?

EVEN WORSE

~~**Must ENGINEER a solution**~~

THIRD OPTION

If it's sketchy, just **skip cleanup step**

Satisfies all project goals!

Solution ~~angst~~ anticlimax

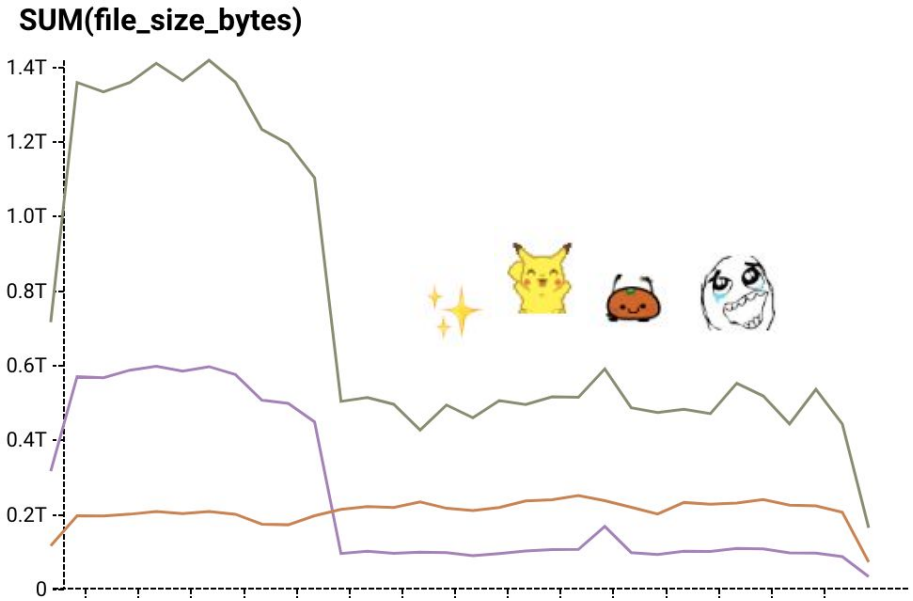
GREAT TRADE: Bypassed deletion 0.01% of time in exchange for peace of mind about fault tolerance

Must **engineer** a solution

How to best satisfy my project **goals**

Solution ~~angst~~ anticlimax

Safely flipped the deletion flag!



thanks :)

rfong@honeycomb.io