# High Performance Cooperative Distributed Systems in Adtech

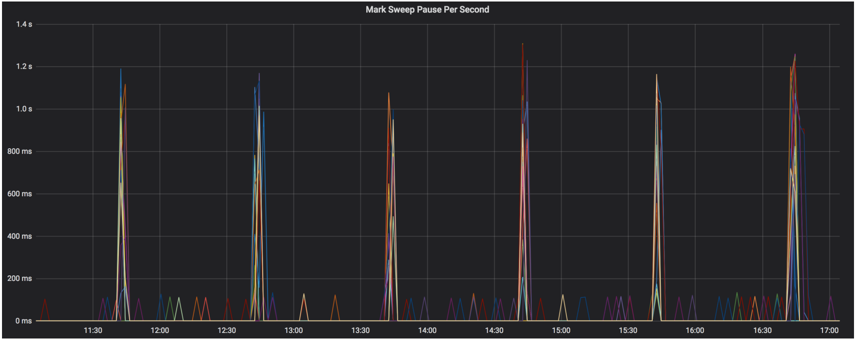
Stan Rosenberg

VP of Engineering
Forensiq
New York, NY

QCon by InfoQ

# Prebid Throughput

# GC Pauses


Mark Sweep Pause Per Second
1.4 s
1.2 s
1.0 s
800 ms
600 ms
400 ms
200 ms
0 ms
11:30
12:00
12:30
13:00
13:30
14:00
14:30
15:00
15:30
16:00
16:30
17:00

# Failure happens all the time

Ken Arnold,

> *When you design distributed systems, you have to say, "Failure happens all the time."*

Fallacies of Distributed Computing (Peter Deutsch),

- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- Transport cost is zero.

QCon by InfoQ

# Past Work

```
public class Comp {
  seq<Comp> children; int total ; // initially total = 1 and children is empty sequence
  Comp parent; // initially parent = null

void add(Comp c)
  requires c ≠ null ∧ c.parent = null;
  requires self ≠ c ∧ I;
  ensures c.parent = self ∧ I;
  effects wr {c}‘parent, {self}‘children;
  effects wr alloc‘total;
{
  assert c ∉ self.children;
  preserves I1({self}) {
    c.parent := self;
    self.children := [c] + self.children;
  }
  self.addToTotal(c.total);
}

int getTotal()
  requires I;
  ensures result = self.total ∧ I;
{
  result := self.total;
}

void addToTotal(int t)
  requires t ≥ 1;
  requires self.total + t =
    1 + sum i; 0 ≤ i < len(self.children) |
      self.children[i].total;
  requires I({self});
  ensures I;
  effects wr alloc‘total;
{
  Comp p; int prv_total;
  p := self;
  while (p ≠ null)
    inv I({p})
    inv p ≠ null ⇒ p.total + t =
      1 + sum i; 0 ≤ i < len(p.children) |
        p.children[i].total;
  {
    assert p.parent ≠ null ⇒
      p ∈ p.parent.children;
    preserves I1({p} + {p.parent}) {
      prv_total := p.total;
      p.total := prv_total + t;
    }
    assert p ≠ p.parent ⇒ p ∉ p.children;
    p := p.parent;
  }
}
```

QCon by InfoQ

# Present Work

```
Exception in thread "main" org.apache.spark.SparkException: Task not serializable
	at org.apache.spark.util.ClosureCleaner$.ensureSerializable(ClosureCleaner.scala:298)
	at org.apache.spark.util.ClosureCleaner$.org$apache$spark$util$ClosureCleaner$$clean(ClosureCleaner.scala:288)
	at org.apache.spark.util.ClosureCleaner$.clean(ClosureCleaner.scala:108)
	at org.apache.spark.SparkContext.clean(SparkContext.scala:2039)
	at org.apache.spark.rdd.RDD$$anonfun$zipPartitions$1.apply(RDD.scala:853)
	at org.apache.spark.rdd.RDD$$anonfun$zipPartitions$1.apply(RDD.scala:853)
	at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:151)
	at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:112)
	at org.apache.spark.rdd.RDD.withScope(RDD.scala:358)
	at org.apache.spark.rdd.RDD.zipPartitions(RDD.scala:852)
	at org.apache.spark.rdd.RDD$$anonfun$zipPartitions$2.apply(RDD.scala:859)
	at org.apache.spark.rdd.RDD$$anonfun$zipPartitions$2.apply(RDD.scala:859)
	at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:151)
	at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:112)
	at org.apache.spark.rdd.RDD.withScope(RDD.scala:358)
	at org.apache.spark.rdd.RDD.zipPartitions(RDD.scala:858)
	at org.apache.spark.sql.execution.WholeStageCodegenExec.doExecute(WholeStageCodegenExec.scala:379)
	at org.apache.spark.sql.execution.SparkPlan$$anonfun$execute$1.apply(SparkPlan.scala:115)
	at org.apache.spark.sql.execution.SparkPlan$$anonfun$execute$1.apply(SparkPlan.scala:115)
	at org.apache.spark.sql.execution.SparkPlan$$anonfun$executeQuery$1.apply(SparkPlan.scala:136)
	at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:151)
	at org.apache.spark.sql.execution.SparkPlan.executeQuery(SparkPlan.scala:133)
	at org.apache.spark.sql.execution.SparkPlan.execute(SparkPlan.scala:114)
	at org.apache.spark.sql.execution.UnionExec$$anonfun$doExecute$1.apply(basicPhysicalOperators.scala:477)
	at org.apache.spark.sql.execution.UnionExec$$anonfun$doExecute$1.apply(basicPhysicalOperators.scala:477)
	at scala.collection.TraversableLike$$anonfun$map$1.apply(TraversableLike.scala:234)
	at scala.collection.TraversableLike$$anonfun$map$1.apply(TraversableLike.scala:234)
	at scala.collection.immutable.List.foreach(List.scala:381)
	at scala.collection.TraversableLike$class.map(TraversableLike.scala:234)
	at scala.collection.immutable.List.map(List.scala:285)
	at org.apache.spark.sql.execution.UnionExec.doExecute(basicPhysicalOperators.scala:477)
	at org.apache.spark.sql.execution.SparkPlan$$anonfun$execute$1.apply(SparkPlan.scala:115)
	at org.apache.spark.sql.execution.SparkPlan$$anonfun$execute$1.apply(SparkPlan.scala:115)
	at org.apache.spark.sql.execution.SparkPlan$$anonfun$executeQuery$1.apply(SparkPlan.scala:136)
	at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:151)
	at org.apache.spark.sql.execution.SparkPlan.executeQuery(SparkPlan.scala:133)
	at org.apache.spark.sql.execution.SparkPlan.execute(SparkPlan.scala:114)
	at org.apache.spark.sql.execution.exchange.ShuffleExchange.prepareShuffleDependency(ShuffleExchange.scala:87)
	at org.apache.spark.sql.execution.exchange.ShuffleExchange$$anonfun$doExecute$1.apply(ShuffleExchange.scala:123)
	at org.apache.spark.sql.execution.exchange.ShuffleExchange$$anonfun$doExecute$1.apply(ShuffleExchange.scala:114)
	at org.apache.spark.sql.catalyst.errors.package$.attachTree(package.scala:52)
	at org.apache.spark.sql.execution.exchange.ShuffleExchange.doExecute(ShuffleExchange.scala:114)
	at org.apache.spark.sql.execution.SparkPlan$$anonfun$execute$1.apply(SparkPlan.scala:115)
	at org.apache.spark.sql.execution.SparkPlan$$anonfun$execute$1.apply(SparkPlan.scala:115)
	at org.apache.spark.sql.execution.SparkPlan$$anonfun$executeQuery$1.apply(SparkPlan.scala:136)
	at org.apache.spark.rdd.RDDOperationScope$.withScope(RDDOperationScope.scala:151)
	at org.apache.spark.sql.execution.SparkPlan.executeQuery(SparkPlan.scala:133)
	at org.apache.spark.sql.execution.SparkPlan.execute(SparkPlan.scala:114)
```

# Intro

- Before, Ph.D., Computer Science; Stevens, Hoboken, 2011
  - Advisor: *David A. Naumann*
  - Dissertation Title: Region Logic: Local Reasoning for Java Programs and its Automation
- Recently, building distributed platforms for startups
  - Appnexus (serving ads faster)
  - PlaceIQ (using location to serve ads)
- VP of Engineering, Forensiq (fighting ad fraud)

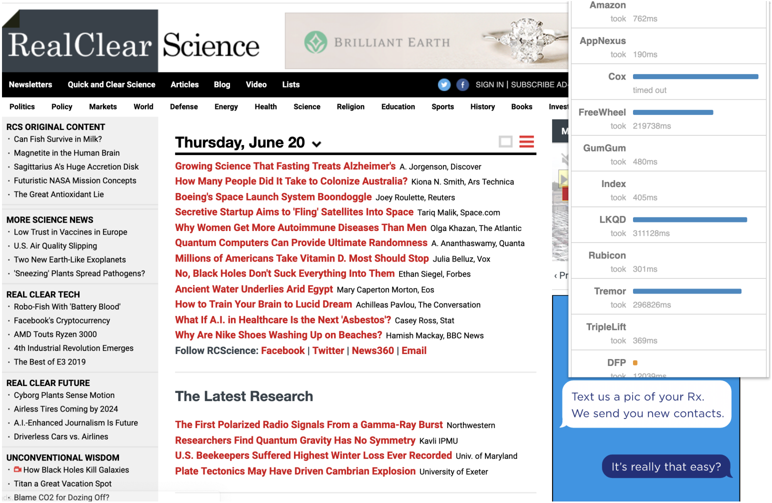
QCon by InfoQ

# Forensiq Overview

forensiq by Impact

- Comprehensive Fraud and Verification SaaS (MRC certified)
- Display Verification (viewability measurements, impression blocking)
- Performance Fraud (stolen attribution, fake action)
- Online scoring via Prebid, Postbid and S2S APIs
- Offline scoring via request log import and reputation lists

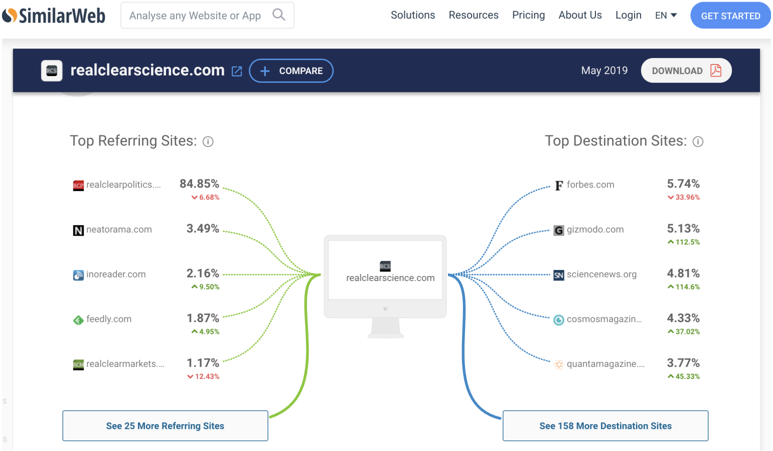
# Fraud Examples



# Fraud Examples

| domain | reason | percent |
|---|---|---|
| realclearscience.com | SIVT | 83.5 |
| realclearscience.com | AUTOMATED_TRAFFIC | 82.7 |
| realclearscience.com | IP_REPUTATION | 14.8 |
| realclearscience.com | PROXY | 13.4 |
| realclearscience.com | GIVT | 0.8 |
| realclearscience.com | HOSTING_PROVIDER | 0.7 |

# Fraud Examples



# Call for Cooperation and Collaboration

Let's improve data quality!

- provide authentic source ip
  - server-side ad-stitching (e.g., AWS Elemental) hides source ip; triggers datacenter traffic
  - MRC notes, "data center traffic is determined to be a consistent source of non-human traffic".
- specify location *type* (OpenRTB 2.5) and *source* to strengthen spoofing detection
- provide campaign/source (aggregate) metrics to help detect client-side JS blocking

# Performance Requirements (Prebid API)

- *high-throughput* – must scale above 1 mil. RPS
- *low-latency* – response p99 $< 10$ms

QCon by InfoQ

# Daily Bid Volume

| | Daily bid request estimates | RTB system |
|---|---|---|
| Index Exchange | 50 billion.[2] | IAB OpenRTB (version unknown)[3] |
| OpenX | 60+ billion.[4] | IAB OpenRTB 2.5[5] |
| Rubicon Project | Unknown billions, daily. Claims to reach 1 billion people's devices.[6] | IAB OpenRTB (version unknown)[7] |
| Oath/AOL | 90 billion.[8] | IAB OpenRTB 2.3[9] |
| AppNexus | 131 billion.[10] | IAB OpenRTB 2.4[11] |
| Smaato | 214 billion.[12] | IAB OpenRTB 2.2, 2.3, 2.4[13] |
| Google DoubleClick | Unknown billions. DoubleClick is the dominant exchange. | IAB OpenRTB 2.2, 2.3, 2.4, 2.5 **and** Authorized Buyers Proto[14] |

$$100 * 10^9 / 86400 \approx 1.1 * 10^6$$

https://fixad.tech/wp-content/uploads/2019/02/4-appendix-on-market-saturation-of-the-systems.pdf

# Common Concerns

| | high-throughput | low-latency |
|---|---|---|
| server backend | ✓ | ✓ |
| KV store | ✓ | ✓ |
| data ingest | ✓ | |
| ETL | ✓ | |
| data pipelines | ✓ | |

- data pipelines
  - Ad Serving: enrichment, budget, attribution, reporting
  - Fraud Detection: enrichment, scoring, reporting

# Guiding Principles

- use NIO
- use compare-and-swap instead of locks (affects OOOE)
- use spatial/temporal locality (prefetch,branch predict)
- minimize coupling and state–keep it simple
- minimize GC pressure
- warmup on startup to trigger JIT
- measure everything with `HdrHistogram`
- benchmark everything with `JMH` and `wrk2`

# Cloud is fast (enough)

- modern hypervisor adds negligible overhead ($< 5\%$)
- consitent performance–"noisy neighbor" is a myth
- networking – 2Gbps per core; up to 32Gbps per VM
  - partitions are infrequent; high inter-region throughput
- local storage – NVMe SSDs; read: 300K IOPS, 2GB/sec
- cloud storage – high-throughput and high-availability
  - strongly consistent (GCS)
  - fast parallel uploads via compose (GCS)

QCon by InfoQ

# Mechanical Sympathy

*Understanding the Hardware Makes You a Better Developer*

- https://mechanical-sympathy.blogspot.com/
- https://dzone.com/articles/mechanical-sympathy
- https://groups.google.com/forum/#!forum/mechanical-sympathy

QCon by InfoQ

# Latency

```
Latency Comparison Numbers
--------------------------
L1 cache reference                           0.5 ns
Branch mispredict                            5   ns
L2 cache reference                           7   ns                      14x L1 cache
Mutex lock/unlock                           25   ns
Main memory reference                      100   ns                      20x L2 cache, 200x L1 cache
Compress 1K bytes with Zippy             3,000   ns        3 us
Send 1K bytes over 1 Gbps network       10,000   ns       10 us
Read 4K randomly from SSD*             150,000   ns      150 us          ~1GB/sec SSD
Read 1 MB sequentially from memory     250,000   ns      250 us
Round trip within same datacenter      500,000   ns      500 us
Read 1 MB sequentially from SSD*     1,000,000   ns    1,000 us    1 ms  ~1GB/sec SSD, 4X memory
Disk seek                           10,000,000   ns   10,000 us   10 ms  20x datacenter roundtrip
Read 1 MB sequentially from disk    20,000,000   ns   20,000 us   20 ms  80x memory, 20X SSD
Send packet CA->Netherlands->CA    150,000,000   ns  150,000 us  150 ms
```

Little's Law: $L = \lambda \times W$, whence throughput is $\propto \frac{1}{\texttt{latency}}$

QCon by InfoQ

# Know Your Data Structures

1000 references to main memory (e.g., linear scan of linked-list) is $\approx$ 100 micros; $(\frac{1}{100}) \times 10^6 = 10,000$ reqs/second

1000 references to L2 cache is $\approx$ 7 micros; $(\frac{1}{7}) \times 10^6 = 142,857$ reqs/second

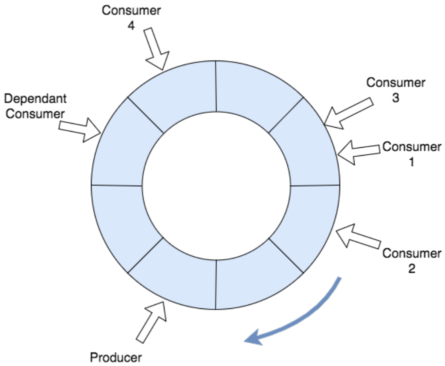
linear search is slower than binary, right?

```
int cnt = 0;
for (int i = 0; i < n; i++)
  cnt += (arr[i] < key);
return cnt < n && arr[cnt] == key;
```

# Disruptor Pattern–Fast Event Processing

- Disruptor is like Java's `BlockingQueue` but *waaaaay* faster!
- RingBuffer
  - one compare-and-swap operation to drain the queue
  - pair of sequence numbers for fast atomic reads/writes
  - exploits speculative racing to eliminate locks
  - consumer message batching results in high-throughput

# Disruptor Pattern




- RingBuffer is pre-allocated (data in `Wrapper.message`)
- compact – `sizeof(disruptor(524,288))` ≈ 14.5MB

# Data Ingest & ETL

- validate each request and apply (payload) limits
- translate JSON to snappy-compressed Avro
- use Disruptor to consume encoded Avro `byte[]`
- append to Avro data file for current 5-min batch
- upload to GCS (throttle to reduce GC pressure)

# Avro & Snappy

16 cores, skylake

java version "1.8.0_202"

@Threads(24), @BenchmarkMode(Mode.Throughput)

| Benchmark | Score | Error | Units |
|---|---|---|---|
| encode | 3741337.244 | ±81494.37 | ops/s |
| encodeCompress | 2699393.673 | ±40130.622 | ops/s |
| decode | 2925509.122 | ±37078.569 | ops/s |
| decodeDecompress | 2771921.410 | ±60483.905 | ops/s |

Also see zstd: `https://facebook.github.io/zstd/`

# Data Ingest & ETL

- early ETL cuts out many downstream inefficiencies
- Avro's performance is on par with Protobuf (also see below)
- throttling uploads and downloads is a must to reduce GC
- eliminate humongous objects (G1)
- naive batching/parallel upload with compose works well
- skip write-ahead log–deal with corrupted Avro blocks

Codegen makes Avro encoder 2x faster: https://github.com/RTBHOUSE/avro-fastserde

QCon by InfoQ

# KV Store–why not Aerospike?

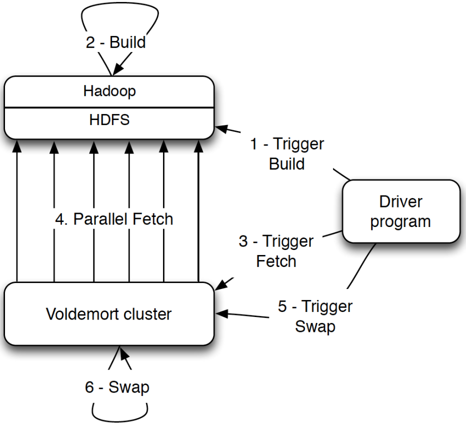
- Pros
  - founded in 2009 (AppNexus was first large deployment)
  - written in C (better resource management in theory)
  - uses Paxos for distributed consensus; heartbeats for node membership
  - supports migrations, rebalancing
  - support cross-datacenter replication
- Cons
  - **No bulk loading**
  - index can get large (RIPEMD is 20 bytes but metadata makes it 64 bytes)
  - log-structured filesystem (copy-on-write); runs compaction in background
  - global 32k bins limit (bins are like column qualifiers)

QCon by InfoQ

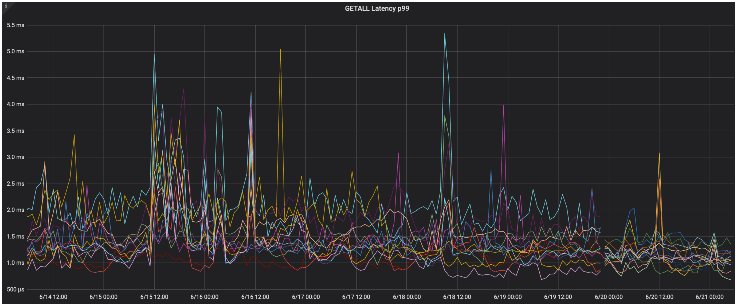
# Low latency KV–Voldemort

- founded in 2009 by LinkedIn (bulk loading main motivator)
- written in Java
- simple get/put API
- uses consistent hashing (similar to Dynamo) to avoid hotspotting
- **bulk loading** and readonly store
- index is compact – uses only 8 bytes of md5(key)
- index file is `mlocked`
- (sort of) supports rebalancing

# Voldemort BuildAndPush



# Voldemort Readonly Performance

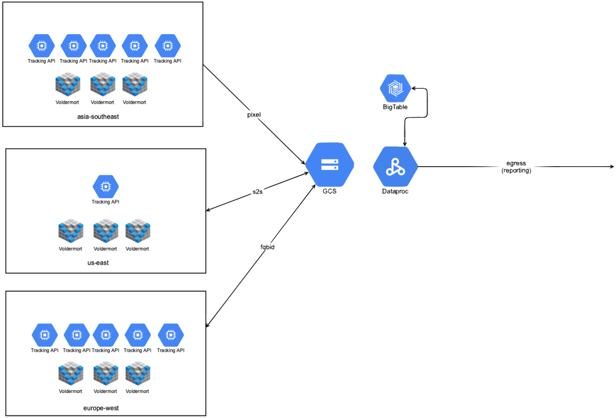


# Custom Voldemort

- added BloomFilter (client-side to reduce RTT)
- added Avro schema versioning
- added Union datastore
- TCP connection pooling is flawed
- reloads create short-lived spikes (hard to pin index)
- 2GB limit per chunk (`ByteBuffer` 32bit *signed* addressing)
- rewrite currently in progress to manage resources more efficiently,
  - rewrite Voldemort backend in `C++`
  - use UDP (potentially with Aeron)
  - use GCS instead of HDFS

QCon by InfoQ

# Putting Things Together



QCon by InfoQ

# Tech. Debt

# Top Two Diseases

- *Legacy* and *Tech. Debt* are the top two diseases of any complex software development
- avoid them at all costs
- Google often rewrites legacy before it's out of control; secondary effect,
    - way of transferring knowledge and ownership to newer team members

Henderson, Fergus. "Software engineering at Google." arXiv preprint arXiv:1702.01715 (2017).

# Rapid Reliable Iteration

- can't iterate quickly without automated verification (i.e., tests)
- invest time into test and benchmarking fixtures early (e.g., write emulators)
- end-to-end (integration) tests, e.g., Selenium, are must-have
- instrument with metrics and measure *everything*
- use design by contract methodology with code reviews

Design by contract was coined by Bertrand Meyer in connection with Eiffel.

# GCP Managed Infrastructure

- distributed, highly available, strongly consistent file system (gcs)
- global *latency-based* load balancing
- zero-downtime rolling deploy
- fast scaling up/down (new instances take $<$ 90 sec. to boot)
- Bigquery (bulk loading, avro/parquet, partitioned tables)
- Bigtable (hbase on steroids)
- syncs (lb logs to bigquery, billing to bigquery, etc.)

https://serverfault.com/questions/881698/random-failed-to-connect-to-backend-errors-on-gce-lb

QCon by InfoQ

# Cloud Tech. is mostly mature

- https://github.com/googleapis/cloud-bigtable-client/issues/1348
- https://github.com/googleapis/google-cloud-java/issues/3531
- https://github.com/googleapis/google-cloud-java/issues/3534
- https://github.com/GoogleCloudPlatform/bigdata-interop/issues/106
- https://github.com/GoogleCloudPlatform/bigdata-interop/issues/153

# Trust but Verify

- cost-effective infrastructure is doable but watch out...
- GCP bait & switch product tactics
  - stackdriver glb logging (free until *insanely* expensive)
  - load-balancer user-defined headers (free until ... )
  - cloud armor (firewall for glb) (free until ... )
- Managed services are black boxes (with limited observability)
  - DNS delegation misconfiguration was $54k over 6 months (no metrics, logging or anomaly detection)
  - dataproc transient failures (no useful logging to determine root cause)
  - dataproc job non-determinstically "stuck" while committing output

QCon by InfoQ

# Summary

- Cloud and OSS is an extremely powerful combination
- High-throughout in Cloud is fairly easy through right design
- Low-latency in Cloud is achievable but takes significantly more effort
  - opportunity to build a managed low-latency KV store
  - storage-as-a-service is still emerging–programmable SSDs
- Fraud is here to stay–cooperation and collaboration with adtech is vital

## Questions?

# EOF