High Performance Cooperative Distributed Systems in Adtech

Stan Rosenberg

VP of Engineering
Forensiq
New York, NY
Prebid Throughput
GC Pauses
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.
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 ∧ l;
        ensures c.parent = self ∧ l;
        effects wr {c}•parent, {self}•children;
        effects wr alloc•total;
    }
        assert c ∉ self.children;
        preserves ll({self}) { }
            c.parent := self;
            self.children := [c] + self.children;
    }
    self.addToTotal(c.total);
}

int getTotal()
    requires l;
    ensures result = self.total ∧ l;
    {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 l({self});
        ensures l;
        effects wr alloc•total;
    }
            Comp p; int prv.total;
            p := self;
            while (p != null)
                inv l({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 ll({p} + {p.parent}) { }
                        prv.total := p.total;
                        p.total := prv.total + t;
                }
                assert p ≠ p.parent ⇒ p ∉ p.children;
                p := p.parent;
    }
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$ensureSerializable(ClosureCleaner.scala:228)
  at org.apache.spark.util.ClosureCleaner$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$withScope(RDDOperationScope.scala:112)
  at org.apache.spark.rdd.RDD.withScope(RDD.scala:385)
  at org.apache.spark.rdd.RDD.zipPartitions(RDD.scala:852)
  at org.apache.spark.rdd.RDD$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$withScope(RDDOperationScope.scala:112)
  at org.apache.spark.rdd.RDD.withScope(RDD.scala:385)
  at org.apache.spark.rdd.RDD.zipPartitions(RDD.scala:858)
  at org.apache.spark.sql.execution.WholeStageCodegenExec.doExecute(WholeStageCodegenExec.scala:379)
  at org.apache.spark.sql.execution.SparkPlan$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$1.apply(SparkPlan.scala:130)
  at org.apache.spark.sql.execution.SparkPlan$executeQuery(SparkPlan.scala:133)
  at org.apache.spark.sql.execution.SparkPlan.execute(SparkPlan.scala:114)
  at org.apache.spark.sql.Union$execute(SparkPlan.scala:477)
  at org.apache.spark.sql.Union$execute(SparkPlan.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.Union$doExecute(Union$doExecute.scala:477)
  at org.apache.spark.sql.SparkPlan$anonfun$execute$1.apply(SparkPlan.scala:115)
  at org.apache.spark.sql.SparkPlan$anonfun$execute$1.apply(SparkPlan.scala:115)
  at org.apache.spark.sql.SparkPlan$executeQuery(SparkPlan.scala:133)
  at org.apache.spark.sql.SparkPlan.execute(SparkPlan.scala:114)
  at org.apache.spark.sql.SparkPlan.execute(SparkPlan.scala:114)
  at org.apache.spark.sql.SparkPlan.execute(SparkPlan.scala:114)
  at org.apache.spark.sql.SparkPlan.execute(SparkPlan.scala:114)
  at org.apache.spark.sql.SparkPlan.execute(SparkPlan.scala:114)
  at org.apache.spark.sql.SparkPlan.execute(SparkPlan.scala:114)
  at org.apache.spark.sql.SparkPlan.execute(SparkPlan.scala:114)
  at org.apache.spark.sql.SparkPlan.execute(SparkPlan.scala:114)
  at org.apache.spark.sql.SparkPlan.execute(SparkPlan.scala:114)
  at org.apache.spark.sql.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)
Forensiq Overview

- 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

Thursday, June 20

Growing Science That Fasting Treats Alzheimer’s
Growing Science That Fasting Treats Alzheimer’s A. Jorgenson, Discover

How Many People Did It Take To Colonize Australia?
How Many People Did It Take To Colonize Australia? Kiona N. Smith, Ars Technica

Boeing’s Space Launch System Boondoggle
Boeing’s Space Launch System Boondoggle Joey Roulette, Reuters

Secretive Startup Aims to ‘Fling’ Satellites Into Space
Secretive Startup Aims to ‘Fling’ Satellites Into Space Tariq Malik, Space.com

Why Women Get More Autoimmune Diseases Than Men
Why Women Get More Autoimmune Diseases Than Men Olga Khazan, The Atlantic

Quantum Computers Can Provide Ultimate Randomness
Quantum Computers Can Provide Ultimate Randomness A. Ananthaswamy, Quanta

Millions of Americans Take Vitamin D. Most Should Stop
Millions of Americans Take Vitamin D. Most Should Stop Julia Belluz, Vox

No, Black Holes Don’t Suck Everything Into Them
No, Black Holes Don’t Suck Everything Into Them Ethan Siegel, Forbes

Ancient Water Underlies Arid Egypt
Ancient Water Underlies Arid Egypt Mary Caperton Morton, Eos

How to Train Your Brain to Lucid Dream
How to Train Your Brain to Lucid Dream Achilles Pavlou, The Conversation

What If A.I. in Healthcare Is the Next Asbestos?
What If A.I. in Healthcare Is the Next Asbestos? Casey Ross, Stat

Why Are Nike Shoes Washing Up on Beaches?
Why Are Nike Shoes Washing Up on Beaches? Hamish Mackay, BBC News

Follow RCS: Facebook | Twitter | News360 | Email

The Latest Research

The First Polarized Radio Signals from a Gamma-Ray Burst
The First Polarized Radio Signals from a Gamma-Ray Burst Northwestern

Researchers Find Quantum Gravity Has No Symmetry
Researchers Find Quantum Gravity Has No Symmetry Kavli IPMU

U.S. Beekeepers Suffered Highest Winter Loss Ever Recorded
U.S. Beekeepers Suffered Highest Winter Loss Ever Recorded Univ. of Maryland

Plate Tectonics May Have Driven Cambrian Explosion
Plate Tectonics May Have Driven Cambrian Explosion University of Exeter
## Fraud Examples

<table>
<thead>
<tr>
<th>domain</th>
<th>reason</th>
<th>percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>realclearscience.com</td>
<td>SIVT</td>
<td>83.5</td>
</tr>
<tr>
<td>realclearscience.com</td>
<td>AUTOMATED_TRAFFIC</td>
<td>82.7</td>
</tr>
<tr>
<td>realclearscience.com</td>
<td>IP_REPUTATION</td>
<td>14.8</td>
</tr>
<tr>
<td>realclearscience.com</td>
<td>PROXY</td>
<td>13.4</td>
</tr>
<tr>
<td>realclearscience.com</td>
<td>GIVT</td>
<td>0.8</td>
</tr>
<tr>
<td>realclearscience.com</td>
<td>HOSTING_PROVIDER</td>
<td>0.7</td>
</tr>
</tbody>
</table>
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 < 10ms
## Daily Bid Volume

<table>
<thead>
<tr>
<th></th>
<th>Daily bid request estimates</th>
<th>RTB system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index Exchange</td>
<td>50 billion.²</td>
<td>IAB OpenRTB (version unknown)³</td>
</tr>
<tr>
<td>OpenX</td>
<td>60+ billion.⁴</td>
<td>IAB OpenRTB 2.5⁵</td>
</tr>
<tr>
<td>Rubicon Project</td>
<td>Unknown billions, daily. Claims to reach 1 billion people’s devices.⁶</td>
<td>IAB OpenRTB (version unknown)⁷</td>
</tr>
<tr>
<td>Oath/AOL</td>
<td>90 billion.⁸</td>
<td>IAB OpenRTB 2.3⁹</td>
</tr>
<tr>
<td>AppNexus</td>
<td>131 billion.¹⁰</td>
<td>IAB OpenRTB 2.4¹¹</td>
</tr>
<tr>
<td>Smaato</td>
<td>214 billion.¹²</td>
<td>IAB OpenRTB 2.2, 2.3, 2.4¹³</td>
</tr>
<tr>
<td>Google DoubleClick</td>
<td>Unknown billions. DoubleClick is the dominant exchange.</td>
<td>IAB OpenRTB 2.2, 2.3, 2.4, 2.5 and Authorized Buyers Proto¹⁴</td>
</tr>
</tbody>
</table>

\[
100 \times 10^9 / 86400 \approx 1.1 \times 10^6
\]

Common Concerns

<table>
<thead>
<tr>
<th></th>
<th>high-throughput</th>
<th>low-latency</th>
</tr>
</thead>
<tbody>
<tr>
<td>server backend</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>KV store</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>data ingest</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>ETL</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>data pipelines</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>

- 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%)
- consistent 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)
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
Latency

Latency Comparison Numbers

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
<td>0.5 ns</td>
<td>14x L1 cache</td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>5 ns</td>
<td></td>
</tr>
<tr>
<td>L2 cache reference</td>
<td>7 ns</td>
<td></td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>25 ns</td>
<td></td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100 ns</td>
<td>20x L2 cache, 200x L1 cache</td>
</tr>
<tr>
<td>Compress 1K bytes with Zippy</td>
<td>3,000 ns</td>
<td>3 us</td>
</tr>
<tr>
<td>Send 1K bytes over 1 Gbps network</td>
<td>10,000 ns</td>
<td>10 us</td>
</tr>
<tr>
<td>Read 4K randomly from SSD*</td>
<td>150,000 ns</td>
<td>150 us</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000 ns</td>
<td>250 us</td>
</tr>
<tr>
<td>Round trip within same datacenter</td>
<td>500,000 ns</td>
<td>500 us</td>
</tr>
<tr>
<td>Read 1 MB sequentially from SSD*</td>
<td>1,000,000 ns</td>
<td>1,000 us</td>
</tr>
<tr>
<td>Disk seek</td>
<td>10,000,000 ns</td>
<td>10 ms</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>20,000,000 ns</td>
<td>20 ms</td>
</tr>
<tr>
<td>Send packet CA-&gt;Netherlands-&gt;CA</td>
<td>150,000,000 ns</td>
<td>150 ms</td>
</tr>
</tbody>
</table>

Little’s Law: $L = \lambda \times W$, whence throughput is $\propto \frac{1}{\text{latency}}$
Know Your Data Structures

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

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

linear search is slower than binary, right?

```c
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 *waaaaaay* 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)) \(\approx 14.5\text{MB}\)
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)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Score</th>
<th>Error</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>encode</td>
<td>3741337.244</td>
<td>±81494.37</td>
<td>ops/s</td>
</tr>
<tr>
<td>encodeCompress</td>
<td>2699393.673</td>
<td>±40130.622</td>
<td>ops/s</td>
</tr>
<tr>
<td>decode</td>
<td>2925509.122</td>
<td>±37078.569</td>
<td>ops/s</td>
</tr>
<tr>
<td>decodeDecompress</td>
<td>2771921.410</td>
<td>±60483.905</td>
<td>ops/s</td>
</tr>
</tbody>
</table>

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
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)
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

1 - Trigger Build
2 - Build
3 - Trigger Fetch
4. Parallel Fetch
5 - Trigger Swap
6 - Swap

Hadoop
HDFS

Driver program

Voldemort cluster
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
Putting Things Together
Tech. Debt

THIS WAY!

WE ARE PROGRESSING SO FAST TOGETHER

THIS DOESN’T LOOK GOOD

TECH DEBT

MONKEYUSER.COM
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

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.)

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
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-deterministically “stuck” while committing output
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