Datadog: A Real-Time Metrics Database for Trillions of Points/Day

Ian NOWLAND (https://twitter.com/inowland)
VP, Metrics and Monitors

Joel BARCIAUSKAS (https://twitter.com/JoelBarciauskas)
Director, Aggregation Metrics

QCon NYC ‘19
Some of Our Customers

- Medium
- monotaro
- MSA
- NASDAQ
- NEXT GAMES
- National Institutes of Health
- Nielsen
- Nikon
- Nokia
- Nutanix
- OneLogin
- PBS
- Peloton
- Philadelphia Phillies
- PriceTravel
- PTC
- Qdoba
- Ricoh
- Roche
- Ferrari
- Systems
- Informatiche
Some of What We Store
Changing Source Lifecycle

Datacenter

Cloud/VM

Containers

Months/years

Seconds
Changing Data Volume

Per User Device

SLIs

Application

System

100's

10,000's
Applying Performance Mantras

- Don't do it
- Do it, but don't do it again
- Do it less
- Do it later
- Do it when they're not looking
- Do it concurrently
- Do it cheaper

*From Craig Hanson and Pat Crain, and the performance engineering community - see http://www.brendangregg.com/methodology.html*
Talk Plan

1. What Are Metrics Databases?
2. Our Architecture
3. Deep Dive On Our Datastores
4. Handling Synchronization
5. Introducing Aggregation
6. Aggregation For Deeper Insights Using Sketches
7. Sketches Enabling Flexible Architecture
Talk Plan

1. What Are Metrics Databases?
2. Our Architecture
3. Deep Dive On Our Datastores
4. Handling Synchronization
5. Introducing Aggregation
6. Aggregation For Deeper Insights Using Sketches
7. Sketches Enabling Flexible Architecture
Example Metrics Query 1

“What is the system load on instance i-xyz across the last 30 minutes”
A Time Series

![Graph showing metric system.load.1 over time, with a value of 0.92 at 11:07:20 and tags including host:i-xyz, env:dev,...]

<table>
<thead>
<tr>
<th>metric</th>
<th>system.load.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
<td>1526382440</td>
</tr>
<tr>
<td>value</td>
<td>0.92</td>
</tr>
<tr>
<td>tags</td>
<td>host:i-xyz, env:dev,...</td>
</tr>
</tbody>
</table>
Example Metrics Query 2

“Alert when the system load, averaged across our fleet in us-east-1a for a 5 minute interval, goes above 90%”
Example Metrics Query 2

“Alert when the system load, averaged across my fleet in us-east-1a for a 5 minute interval, goes above 90%”
Metrics Name and Tags

**Name:** single string defining what you are measuring, e.g.

- system.cpu.user
- aws.elb.latency
- dd.frontend.internal.ajax.queue.length.total

**Tags:** list of k:v strings, used to qualify metric and add dimensions to filter/aggregate over, e.g.

- ['host:server-1', 'availability-zone:us-east-1a', 'kernel_version:4.4.0']
- ['host:server-2', 'availability-zone:us-east-1a', 'kernel_version:2.6.32']
- ['host:server-3', 'availability-zone:us-east-1b', 'kernel_version:2.6.32']
Tags for all the dimensions

Host / container: system metrics by host
Application: internal cache hit rates, timers by module
Service: hits, latencies or errors/s by path and/or response code
Business: # of orders processed, $'s per second by customer ID
Talk Plan

1. What Are Metrics Databases?
2. Our Architecture
3. Deep Dive On Our Datastores
4. Handling Synchronization
5. Introducing Aggregation
6. Aggregation For Deeper Insights Using Sketches
7. Sketches Enabling Flexible Architecture
Pipeline Architecture

- Metrics sources
- Slack/Email/PagerDuty etc
- Customer Browser

- Intake
- Monitors and Alerts
- Web frontend & APIs
- Query System

Data Stores
Performance mantras

• Don't do it
• **Do it, but don't do it again**
• Do it less
• Do it later
• Do it when they're not looking
• Do it concurrently
• Do it cheaper
Performance mantras

- Don't do it
- **Do it, but don't do it again - query caching**
- Do it less
- Do it later
- Do it when they're not looking
- Do it concurrently
- Do it cheaper
Pipeline Architecture

Metrics sources
Slack/Email/PagerDuty etc
Customer Browser

Customer

Intake

Monitors and Alerts
Web frontend & APIs

Query System

Data Stores

Query Cache
Pipeline Architecture

Metrics sources

Slack/Email/PagerDuty etc

Customer Browser

Customer

Intake

Data Stores

Query System

Query Cache

Monitors and Alerts

Web frontend & APIs

Data Stores
Metrics Store Characteristics

- Most metrics report with a tag set for quite some time
  => Therefore separate tag stores from time series stores
Pipeline Architecture

- Metrics sources
- Slack/Email/PagerDuty etc
- Customer Browser
- Intake
- Data Stores
- Monitors and Alerts
- Web frontend & APIs
- Query System
- Query Cache

Customer
Kafka for Independent Storage Systems

Incoming Data -> Intake -> Kafka Points -> Kafka Tag Sets

Kafka Points -> Tag Sets

Tag Sets -> Store 1

Tag Sets -> Store 2

Store 1 -> Tag Index

Store 2 -> Tag Index

Tag Index -> Tag Describer

Tag Describer -> S3 Writer

S3 Writer -> Query System

Query System -> S3

S3 -> Outgoing Data
Performance mantras

- Don't do it
- Do it, but don't do it again - query caching
- Do it less
- **Do it later - minimize processing on path to persistence**
- Do it when they're not looking
- Do it concurrently
- Do it cheaper
Kafka for Independent Storage Systems

Incoming Data → Intake → Kafka Points → Kafka Tag Sets → S3 Writer → Tag Index → Tag Descriptor → Store 1 → Store 2 → Query System → S3 → Outgoing Data
Scaling through Kafka

Data is separated by **partition** to distribute it.
Partitions are customers, or a mod hash of their metric name.
This also gives us **isolation**.
Performance mantras

- Don't do it
- Do it, but don't do it again - query caching
- Do it less
- Do it later - minimize processing on path to persistence
- Do it when they're not looking
- **Do it concurrently - use independent horizontally scalable data stores**
- Do it cheaper
Talk Plan

1. What Are Metrics Databases?
2. Our Architecture
3. Deep Dive On Our Datastores
4. Handling Synchronization
5. Introducing Aggregation
6. Aggregation For Deeper Insights Using Sketches
7. Sketches Enabling Flexible Architecture
## Per Customer Volume Ballparking

| $10^4$ | Number of apps; 1,000’s hosts times 10’s containers |
| $10^3$ | Number of metrics emitted from each app/container |
| $10^0$ | 1 point a second per metric |
| $10^5$ | Seconds in a day (actually 86,400) |
| $10^1$ | Bytes/point (8 byte float, amortized tags) |

$= 10^{13}$ 10 Terabytes a Day For One Average Customer
Volume Math

- $210 to store 10 TB in S3 for a month
- $60,000 for a month rolling queryable (300 TB)
- But S3 is not for real time, high throughput queries
## Cloud Storage Characteristics

<table>
<thead>
<tr>
<th>Type</th>
<th>Max Capacity</th>
<th>Bandwidth</th>
<th>Latency</th>
<th>Cost/TB for 1 month</th>
<th>Volatility</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM¹</td>
<td>4 TB</td>
<td>80 GB/s</td>
<td>0.08 us</td>
<td>$1,000</td>
<td>Instance Reboot</td>
</tr>
<tr>
<td>SSD²</td>
<td>60 TB</td>
<td>12 GB/s</td>
<td>1 us</td>
<td>$60</td>
<td>Instance Failures</td>
</tr>
<tr>
<td>EBS io1</td>
<td>432 TB</td>
<td>12 GB/s</td>
<td>40 us</td>
<td>$400</td>
<td>Data Center Failures</td>
</tr>
<tr>
<td>S3</td>
<td>Infinite</td>
<td>12 GB/s³</td>
<td>100+ ms</td>
<td>$21⁴</td>
<td>11 nines durability</td>
</tr>
<tr>
<td>Glacier</td>
<td>Infinite</td>
<td>12 GB/s³</td>
<td>hours</td>
<td>$4⁴</td>
<td>11 nines durability</td>
</tr>
</tbody>
</table>

1. X1e.32xlarge, 3 year non convertible, no upfront reserved instance
2. i3en.24xlarge, 3 year non convertible, no upfront reserved instance
3. Assumes can highly parallelize to load network card of 100Gbps instance type. Likely does not scale out.
4. Storage Cost only
Volume Math

- 80 x1e.32xlarge DRAM for a month
- $300,000 to store for a month
- This is with no indexes or overhead
- And people want to query much more than a month.
Performance mantras

• Don't do it
• Do it, but don't do it again - query caching
  • **Do it less - only index what you need**
• Do it later - minimize processing on path to persistence
• Do it when they're not looking
• Do it concurrently - use independent horizontally scalable data stores
• Do it cheaper
Returning to an Example Query

“Alert when the system load, averaged across our fleet in us-east-1a for a 5 minute interval, goes above 90%”
# Queries We Need to Support

<table>
<thead>
<tr>
<th>Query Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIBE TAGS</td>
<td>What tags are queryable for this metric?</td>
</tr>
<tr>
<td>TAG INDEX</td>
<td>Given a time series id, what tags were used?</td>
</tr>
<tr>
<td>TAG INVERTED INDEX</td>
<td>Given some tags and a time range, what were the time series ingested?</td>
</tr>
<tr>
<td>POINT STORE</td>
<td>What are the values of a time series between two times?</td>
</tr>
</tbody>
</table>
Performance mantras

- Don't do it
- Do it, but don't do it again - query caching
- Do it less - only index what you need
- Do it later - minimize processing on path to persistence
- Do it when they're not looking
- Do it concurrently - use independent horizontally scalable data stores
- Do it cheaper
Performance mantras

- Don't do it
- Do it, but don't do it again - query caching
- Do it less - only index what you need
- Do it later - minimize processing on path to persistence
- Do it when they're not looking
- Do it concurrently - use independent horizontally scalable data stores
- Do it cheaper - use hybrid data storage types and technologies
# Cloud Storage Characteristics

<table>
<thead>
<tr>
<th>Type</th>
<th>Max Capacity</th>
<th>Bandwidth</th>
<th>Latency</th>
<th>Cost/TB for 1 month</th>
<th>Volatility</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM&lt;sup&gt;1&lt;/sup&gt;</td>
<td>4 TB</td>
<td>80 GB/s</td>
<td>0.08 us</td>
<td>$1,000</td>
<td>Instance Reboot</td>
</tr>
<tr>
<td>SSD&lt;sup&gt;2&lt;/sup&gt;</td>
<td>60 TB</td>
<td>12 GB/s</td>
<td>1 us</td>
<td>$60</td>
<td>Instance Failures</td>
</tr>
<tr>
<td>EBS io1</td>
<td>432 TB</td>
<td>12 GB/s</td>
<td>40 us</td>
<td>$400</td>
<td>Data Center Failures</td>
</tr>
<tr>
<td>S3</td>
<td>Infinite</td>
<td>12 GB/s&lt;sup&gt;3&lt;/sup&gt;</td>
<td>100+ ms</td>
<td>$21&lt;sup&gt;4&lt;/sup&gt;</td>
<td>11 nines durability</td>
</tr>
<tr>
<td>Glacier</td>
<td>Infinite</td>
<td>12 GB/s&lt;sup&gt;3&lt;/sup&gt;</td>
<td>hours</td>
<td>$4&lt;sup&gt;4&lt;/sup&gt;</td>
<td>11 nines durability</td>
</tr>
</tbody>
</table>

1. X1e.32xlarge, 3 year non convertible, no upfront reserved instance
2. i3en.24xlarge, 3 year non convertible, no upfront reserved instance
3. Assumes can highly parallelize to load network card of 100Gbps instance type. Likely does not scale out.
4. Storage Cost only
## Cloud Storage Characteristics

<table>
<thead>
<tr>
<th>Type</th>
<th>Max Capacity</th>
<th>Bandwidth</th>
<th>Latency</th>
<th>Cost/TB for 1 month</th>
<th>Volatility</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAM(^1)</td>
<td>4 TB</td>
<td>80 GB/s</td>
<td>0.08 us</td>
<td>$1,000</td>
<td>Instance Reboot</td>
</tr>
<tr>
<td>SSD(^2)</td>
<td>60 TB</td>
<td>12 GB/s</td>
<td>1 us</td>
<td>$60</td>
<td>Instance Failures</td>
</tr>
<tr>
<td>EBS io1</td>
<td>432 TB</td>
<td>12 GB/s</td>
<td>40 us</td>
<td>$400</td>
<td>Data Center Failures</td>
</tr>
<tr>
<td>S3</td>
<td>Infinite</td>
<td>12 GB/s(^3)</td>
<td>100+ ms</td>
<td>$21(^4)</td>
<td>11 nines durability</td>
</tr>
<tr>
<td>Glacier</td>
<td>Infinite</td>
<td>12 GB/s(^3)</td>
<td>hours</td>
<td>$4(^4)</td>
<td>11 nines durability</td>
</tr>
</tbody>
</table>

1. X1e.32xlarge, 3 year non convertible, no upfront reserved instance
2. i3en.24xlarge, 3 year non convertible, no upfront reserved instance
3. Assumes can highly parallelize to load network card of 100Gbps instance type. Likely does not scale out.
4. Storage Cost only
## Hybrid Data Storage Types

<table>
<thead>
<tr>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIBE TAGS</td>
</tr>
<tr>
<td>TAG INDEX</td>
</tr>
<tr>
<td>TAG INVERTED INDEX</td>
</tr>
<tr>
<td>POINT STORE</td>
</tr>
<tr>
<td>QUERY RESULTS</td>
</tr>
</tbody>
</table>
## Hybrid Data Storage Types

<table>
<thead>
<tr>
<th>System</th>
<th>Type</th>
<th>Persistence</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIBE TAGS</td>
<td>Local SSD</td>
<td>Years</td>
</tr>
<tr>
<td>TAG INDEX</td>
<td>DRAM</td>
<td>Cache (Hours)</td>
</tr>
<tr>
<td></td>
<td>Local SSD</td>
<td>Years</td>
</tr>
<tr>
<td>TAG INVERTED INDEX</td>
<td>DRAM</td>
<td>Hours</td>
</tr>
<tr>
<td></td>
<td>On SSD</td>
<td>Days</td>
</tr>
<tr>
<td></td>
<td>S3</td>
<td>Years</td>
</tr>
<tr>
<td>POINT STORE</td>
<td>DRAM</td>
<td>Hours</td>
</tr>
<tr>
<td></td>
<td>Local SSD</td>
<td>Days</td>
</tr>
<tr>
<td></td>
<td>S3</td>
<td>Years</td>
</tr>
<tr>
<td>QUERY RESULTS</td>
<td>DRAM</td>
<td>Cache (Days)</td>
</tr>
</tbody>
</table>
## Hybrid Data Storage Technologies

<table>
<thead>
<tr>
<th>System</th>
<th>Type</th>
<th>Persistence</th>
<th>Technology</th>
<th>Why?</th>
</tr>
</thead>
<tbody>
<tr>
<td>DESCRIBE TAGS</td>
<td>Local SSD</td>
<td>Years</td>
<td>LevelDB</td>
<td>High performing single node k,v</td>
</tr>
<tr>
<td>TAG INDEX</td>
<td>DRAM</td>
<td>Cache (Hours)</td>
<td>Redis</td>
<td>Very high performance, in memory k,v</td>
</tr>
<tr>
<td></td>
<td>Local SSD</td>
<td>Years</td>
<td>Cassandra</td>
<td>Horizontal scaling, persistent k,v</td>
</tr>
<tr>
<td>TAG INVERTED INDEX</td>
<td>DRAM</td>
<td>Hours</td>
<td>In house</td>
<td>Very customized index data structures</td>
</tr>
<tr>
<td></td>
<td>On SSD</td>
<td>Days</td>
<td>RocksDB + SQLite</td>
<td>Rich and flexible queries</td>
</tr>
<tr>
<td></td>
<td>S3</td>
<td>Years</td>
<td>Parquet</td>
<td>Flexible Schema over time</td>
</tr>
<tr>
<td>POINT STORE</td>
<td>DRAM</td>
<td>Hours</td>
<td>In house</td>
<td>Very customized index data structures</td>
</tr>
<tr>
<td></td>
<td>Local SSD</td>
<td>Days</td>
<td>In house</td>
<td>Very customized index data structures</td>
</tr>
<tr>
<td></td>
<td>S3</td>
<td>Years</td>
<td>Parquet</td>
<td>Flexible Schema over time</td>
</tr>
<tr>
<td>QUERY RESULTS</td>
<td>DRAM</td>
<td>Cache (Days)</td>
<td>Redis</td>
<td>Very high performance, in memory k,v</td>
</tr>
</tbody>
</table>
Talk Plan

1. What Are Metrics Databases?
2. Our Architecture
3. Deep Dive On Our Datastores
4. **Handling Synchronization**
5. Introducing Aggregation
6. Aggregation For Deeper Insights Using Sketches
7. Sketches Enabling Flexible Architecture
Alerts/Monitors Synchronization

- Level sensitive
  - False positives is almost as important as false negative
- Small delay preferable to evaluating incomplete data
- Synchronization need is to be sure evaluation bucket is filled before processing
Pipeline Architecture

- Metrics sources
- Slack/Email/PagerDuty etc
- Customer Browser

Intake

Monitors and Alerts

Web frontend & APIs

Query System

Query Cache

Data Stores

Inject heartbeat here
Pipeline Architecture

- Metrics sources
- Slack/Email/PagerDuty etc
- Customer Browser

Intake

- Monitors and Alerts
- Web frontend & APIs

Query System

Data Stores

Query Cache

Customer

- Inject heartbeat here

And test it gets to here
Heartbeats for Synchronization

Semantics:
- 1 second tick time for metrics
- Last write wins to handle agent concurrency
- Inject fake data as heartbeat through pipeline

Then:
- Monitor evaluator ensure heartbeat gets through before evaluating next period

Challenges:
- With sharding and multiple stores, lots of independent paths to make sure heartbeats go through
Performance mantras

- Don't do it - build the minimal synchronization needed
- Do it, but don't do it again - query caching
- Do it less - only index what you need
- Do it later - minimize processing on path to persistence
- Do it when they're not looking
- Do it concurrently - use independent horizontally scalable data stores
- Do it cheaper - use hybrid data storage types and technologies
Talk Plan

1. What Are Metrics Databases?
2. Our Architecture
3. Deep Dive On Our Datastores
4. Handling Synchronization
5. Introducing Aggregation
6. Aggregation For Deeper Insights Using Sketches
7. Sketches Enabling Flexible Architecture
Types of metrics

Counter, aggregate by sum
Ex: Requests, errors/s, total time spent (stopwatch)

Gauges, aggregate by last or avg
Ex: CPU/network/disk use, queue length
Aggregation

<table>
<thead>
<tr>
<th>$t_0$</th>
<th>$t_1$</th>
<th>$t_2$</th>
<th>$t_3$</th>
<th>$t_4$</th>
<th>$t_5$</th>
<th>$t_6$</th>
<th>$t_7$</th>
<th>$t_8$</th>
<th>$t_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 1, 0, 1, 0, 1, 0, 1, 0, 1</td>
<td>0, 1, 2, 3, 4, 5, 6, 7, 8, 9</td>
<td>5, 5, 5, 5, 5, 5, 5, 5, 5, 5</td>
<td>0, 2, 4, 8, 16, 32, 64, 128, 256, 512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Query output

Counters: \{5, 40, 50, 1023\}

Gauges (average): \{0.5, 4, 5, 102.3\}

Gauges (last): \{1, 9, 5, 512\}
Query characteristics

User:

- Bursty and unpredictable
- Latency Sensitive - ideal end user response is 100ms, 1s at most.
- Skews to recent data, but want same latency on old data
Query characteristics

Dashboards:

- Predictable
- Important enough to save
- Looking for step-function changes, e.g. performance regressions, changes in usage
Focus on outputs

These graphs are *both* aggregating 70k series
Not a lot, but still output 10x to 2000x less than input!
Performance mantras

- Don't do it - build the minimal synchronization needed
- Do it, but don't do it again - query caching
- Do it less - only index what you need
- Do it later - minimize processing on path to persistence
- **Do it when they're not looking?**
- Do it concurrently - use independent horizontally scalable data stores
- Do it cheaper - use hybrid data storage types and technologies
Pipeline Architecture

Metrics sources
Slack/Email/PagerDuty etc
Customer Browser

Aggregation Points

Intake
Query System
Query Cache

Data Stores

Monitors and Alerts
Web frontend & APIs
Pipeline Architecture

Metrics sources
Slack/Email/PagerDuty etc
Customer Browser

Customer

Aggregation Points

Intake
Query System
Web frontend & APIs
Monitors and Alerts

Query Cache
Data Stores
Streaming Aggregator
Pipeline Architecture

Metrics sources
Slack/Email/PagerDuty etc
Customer Browser

Aggregation Points

Intake
Monitors and Alerts
Web frontend & APIs
Query System
Query Cache
Data Stores
Streaming Aggregator

Customer

No one's looking here!
Performance mantras

- Don't do it - build the minimal synchronization needed
- Do it, but don't do it again - query caching
- Do it less - only index what you need
- Do it later - minimize processing on path to persistence
- Do it when they're not looking - pre-aggregate
- Do it concurrently - use independent horizontally scalable data stores
- Do it cheaper - use hybrid data storage types and technologies
Talk Plan

1. What Are Metrics Databases?
2. Our Architecture
3. Deep Dive On Our Datastores
4. Handling Synchronization
5. Introducing Aggregation
6. **Aggregation For Deeper Insights Using Sketches**
7. Sketches Enabling Flexible Architecture
Distributions

Aggregate by percentile or SLO
(count of values above or below a threshold)

Ex: Latency, request size
Calculating distributions

<table>
<thead>
<tr>
<th>t₀</th>
<th>t₁</th>
<th>t₂</th>
<th>t₃</th>
<th>t₄</th>
<th>t₅</th>
<th>t₆</th>
<th>t₇</th>
<th>t₈</th>
<th>t₉</th>
</tr>
</thead>
</table>

{0, 1, 0, 1, 0, 1, 0, 1, 0, 1}

{0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

{5, 5, 5, 5, 5, 5, 5, 5, 5, 5}

{0, 2, 4, 8, 16, 32, 64, 128, 256, 512}

{0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 2, 2, 3, 4, 4, 5, 5, 5, 5, 5, 5, 6, 7, 8, 8, 9, 16, 32, 64, 128, 256, 512}
Performance mantras

- Don't do it - build the minimal synchronization needed
- Do it, but don't do it again - query caching
- Do it less - only index what you need
- Do it later - minimize processing on path to persistence
- Do it when they're not looking - pre-aggregate
- Do it concurrently - use independent horizontally scalable data stores
- **Do it cheaper again?**
What are "sketches"?

Data structures designed for operating on streams of data

- Examine each item a limited number of times (ideally once)
- Limited memory usage (logarithmic to the size of the stream, or fixed)
Examples of sketches

HyperLogLog

- Cardinality / unique count estimation
- Used in Redis PFADD, PFCOUNT, PFMERGE

Others: Bloom filters (also for set membership), frequency sketches (top-N lists)
Tradeoffs

Understand the tradeoffs - speed, accuracy, space

What other characteristics do you need?

• Well-defined or arbitrary range of inputs?
• What kinds of queries are you answering?
Approximation for distribution metrics

What's important for approximating distribution metrics?

- Bounded error
- Performance - size, speed of inserts
- Aggregation (aka "merging")
How do you compress a distribution
Histograms

Basic example from OpenMetrics / Prometheus

```
# HELP http_request_duration_seconds A histogram of the request duration.
# TYPE http_request_duration_seconds histogram
http_request_duration_seconds_bucket{le="0.05"} 24054
http_request_duration_seconds_bucket{le="0.1"} 33444
http_request_duration_seconds_bucket{le="0.2"} 100392
http_request_duration_seconds_bucket{le="0.5"} 129389
http_request_duration_seconds_bucket{le="1"} 133988
http_request_duration_seconds_bucket{le="+Inf"} 144320
http_request_duration_seconds_sum 53423
http_request_duration_seconds_count 144320
```
Histograms

Basic example from OpenMetrics / Prometheus

<table>
<thead>
<tr>
<th>Time spent</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= 0.05 (50ms)</td>
<td>24054</td>
</tr>
<tr>
<td>&lt;= 0.1 (100ms)</td>
<td>33444</td>
</tr>
<tr>
<td>&lt;= 0.2 (200ms)</td>
<td>100392</td>
</tr>
<tr>
<td>&lt;= 0.5 (500ms)</td>
<td>129389</td>
</tr>
<tr>
<td>&lt;= 1s</td>
<td>133988</td>
</tr>
<tr>
<td>&gt; 1s</td>
<td>144320</td>
</tr>
</tbody>
</table>

median = ~158ms (using linear interpolation)

p99 = ?!
Rank and relative error
Rank and relative error
Relative error

In metrics, specifically latency metrics, we care about both the distribution of data as well as **specific values**

E.g., for an SLO, I want to know, is my p99 500ms or less?
Relative error bounds mean we can answer this: Yes, within 99% of requests are $\leq 500\text{ms} \pm 1\%
Otherwise stated: 99% of requests are **guaranteed** $\leq 505\text{ms}$
Fast insertion

Each insertion is just two operations - find the bucket, increase the count (sometimes there's an allocation)
Fixed Size - how?

With certain distributions, we may reach the maximum number of buckets (in our case, 4000)

- Roll up lower buckets - lower percentiles are generally not as interesting!*

*Note that we've yet to find a data set that actually needs this in practice
Aggregation and merging

"a binary operation is **commutative** if *changing the order* of the operands does not change the result"

Why is this important?
Talk Plan

1. What Are Metrics Databases?
2. Our Architecture
3. Deep Dive On Our Datastores
4. Handling Synchronization
5. Introducing Aggregation
6. Aggregation For Deeper Insights Using Sketches
7. Sketches Enabling Flexible Architecture
Before, during, save for later

If we have two-way mergeable sketches, we can re-aggregate the aggregations

• Agent
• Streaming during ingestion
• At query time
• In the data store (saving partial results)
Pipeline Architecture

Metrics sources

Slack/Email/PagerDuty etc

Customer Browser

Intake

Query System

Streaming Aggregator

Data Stores

Query Cache

Monitoring and Alerts

Web frontend & APIs

Aggregation Points

Customer
DDSketch

DDSketch (Distributed Distribution Sketch) is open source (part of the agent today)

- Presenting at VLDB2019 in August
- Open-sourcing standalone versions in several languages
Performance mantras

- Don't do it - build the minimal synchronization needed
- Do it, but don't do it again - query caching
- Do it less - only index what you need
- Do it later - minimize processing on path to persistence
- Do it when they're not looking - pre-aggregate
- Do it concurrently - use independent horizontally scalable data stores
- **Do it cheaper - use hybrid data storage types and technologies**
Performance mantras

- Don't do it - build the minimal synchronization needed
- Do it, but don't do it again - query caching
- Do it less - only index what you need
- Do it later - minimize processing on path to persistence
- Do it when they're not looking - pre-aggregate
- Do it concurrently - use independent horizontally scalable data stores
- **Do it cheaper - use hybrid data storage types and technologies, and use compression techniques based on what customers really need**
Summary

- **Don't do it** - build the bare minimal synchronization needed
- **Do it, but don't do it again** - use query caching
- **Do it less** - only index what you need
- **Do it later** - minimize processing on path to persistence
- **Do it when they're not looking** - pre-aggregate where is cost effective
- **Do it concurrently** - use independent horizontally scaleable data stores
- **Do it cheaper** - use hybrid data storage types and technologies, and use compression techniques based on what customers really need
Thank You
Challenges and opportunities of aggregation

• Challenges:
  • Accuracy
  • Latency

• Opportunity:
  • Orders of magnitude performance improvement on common and highly visible queries
Human factors and dashboards

- Human-latency sensitive - high visibility
  Late-arriving data makes people nervous
- Human granularity - how many lines can you reason about on a dashboard?

Oh no...
Where aggregation happens

At the metric source (agent/lambda/etc)
- Counts by sum
- Gauges by last

At query time
- Arbitrary user selection (avg/sum/min/max)
- Impacts user experience
Adding a new metric type

Counters, gauges, **distributions**!

Used gauges for latency, etc, but aggregate by last is not what you want

Need to update the agent, libraries, integrations

We're learning and building on what we have today
Building blocks

We have a way to move data around (Kafka)
We have ways to index that data (tagsets)
We know **how** to separate recent and historical data
Plan for the **future**

[Lego / puzzle with gaps]
Connect the dots