Processing Big Data in Motion
Streaming Data Ingestion and Processing

Roger Barga
General Manager
Kinesis Streaming Services, AWS

June 24th, 2016
Interest in and demand for stream data processing is rapidly increasing* ...

* Understatement of the year (credit to Kostas Tzoumas)...
Most data is produced continuously.

Metering Record

```json
{
  "payerId": "Joe",
  "productCode": "AmazonS3",
  "clientProductCode": "AmazonS3",
  "usageType": "Bandwidth",
  "operation": "PUT",
  "value": "22490",
  "timestamp": "1216674828"
}
```

Common Log Entry

```
```

NASDAQ OMX Record

```
<R,AMZN,T,G,R1>
```

Beacons

```
"SeattlePublicWater/Kinesis/123/Realtime" – 412309129140
```

MQTT Record

```
<165>1 2003-10-11T22:14:15.003Z mymachine.example.com
```

Health Monitors

```
SMART BUILDINGS
```

Syslog Entry
Time is money...

**Recent data** is highly valuable
- If you act on it in time
- Perishable Insights (M. Gualtieri, Forrester)

**Old + Recent** data is more valuable
- If you have the means to combine them
Most ‘big data’ (Hadoop) jobs process data that was continuously generated.

Foundational for business.

Enable new class of applications that process data continuously.
Agenda

- **Scalable & Durable Data Ingest**
  - A quick word on our motivation
  - Kinesis Streams, through a simple example

- **Continuous Stream Data Processing**
  - Kinesis Client Library (KCL)
  - How customers are using Kinesis Streams today

- **Building on Kinesis Streams**
  - Kinesis Firehose
  - Kinesis Analytics
Our Motivation for Continuous Processing

AWS Metering service

• 100s of millions of billing records per second
• Terabytes++ per hour
• Hundreds of thousands of sources
• For each customer: gather all metering records & compute monthly bill
• Auditors guarantee 100% accuracy at months end

Seem perfectly reasonable to run as a batch, but relentless pressure for realtime…

With a Data Warehouse to load

• 1000s extract-transform-load (ETL) jobs every day
• Hundreds of thousands of files per load cycle
• Thousands of daily users, hundreds of queries per hour
Our Motivation for Continuous Processing

AWS Metering service

- 100s of millions of billing records per second
- Terabytes++ per hour
- Hundreds of thousands of sources
- For each customer: gather all metering records & compute monthly bill
- Auditors guarantee 100% accuracy at months end

Other Service Teams, Similar Requirements

- CloudWatch Logs and CloudWatch Metrics
- CloudFront API logging
- ‘Snitch’ internal datacenter hardware metrics
Right Tool for the Job

Enable Streaming **Data Ingestion** and **Processing**

**Real-time Ingest**
- Highly Scalable
- Durable
- Replayable Reads

**Continuous Processing**
- Support multiple simultaneous data processing applications
- Load-balancing incoming streams, scale out processing
- Fault-tolerance, Checkpoint / Replay

**Amazon Kinesis**
Example application
twitter-trends.com website
Too big to handle on one box
The solution: streaming map/reduce
Core concepts

- Data record
- Stream
- Partition key
- Shard
- Worker
- Global top-10
- My top-10
- twitter-trends.com
- Shard: 14, 17, 18, 21, 23
- Sequence number
How this relates to Kinesis
Kinesis Streaming Data Ingestion

• Streams are made of **Shards**
• Each Shard ingests data up to 1MB/sec, and up to 1000 TPS
• Producers use a PUT call to store data in a Stream: PutRecord {Data, PartitionKey, StreamName}
• Each Shard emits up to 2 MB/sec
• All data is stored for **24 hours, 7 days** if extended retention is ‘ON’
• **Scale** Kinesis streams by adding or removing Shards
• **Replay** data from retention period
Real-Time Streaming Data Ingestion

- Inexpensive: $0.014 per 1,000,000 PUT Payload Units

- Durable, highly consistent storage replicates data across three data centers (availability zones)

- Millions of sources producing 100s of terabytes per hour

- 25 – 40ms

- 100 – 150ms
Kinesis Client Library
Using the Kinesis API directly

```java
KinesisIterator = getShardIterator(shardId, LATEST);

while (true) {
    [records, iterator] = getNextRecords(iterator, maxRecsToReturn);
    process(records);
}

process(records): {
    for (record in records) {
        updateLocalTop10(record);
    }
    if (timeToDoOutput()) {
        writeLocalTop10ToDDB();
    }
}

while (true) {
    localTop10Lists = scanDDBTable();
    updateGlobalTop10List(localTop10Lists);
    sleep(10);
}
```
Challenges with using the Kinesis API directly

Manual creation of workers and assignment to shards?

How many EC2 instances?

How many workers per EC2 instance?

twitter-trends.com

Kinesis application
Using the Kinesis Client Library
Elasticity and Load Balancing
Fault Tolerance Support

KINESIS

Availability Zone 1

Availability Zone 3

Shard mgmt table

twitter-trends.com

Amazon Web Services
Worker Fail Over

<table>
<thead>
<tr>
<th>LeaseKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-0</td>
<td>Worker1</td>
<td>85</td>
</tr>
<tr>
<td>Shard-1</td>
<td>Worker2</td>
<td>94</td>
</tr>
<tr>
<td>Shard-2</td>
<td>Worker3</td>
<td>76</td>
</tr>
</tbody>
</table>
Worker Fail Over

<table>
<thead>
<tr>
<th>Shard</th>
<th>Worker</th>
<th>LeaseKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-0</td>
<td>Worker1</td>
<td></td>
<td>Worker1</td>
<td>85 86</td>
</tr>
<tr>
<td>Shard-1</td>
<td>Worker2</td>
<td></td>
<td>Worker2</td>
<td>94</td>
</tr>
<tr>
<td>Shard-2</td>
<td>Worker3</td>
<td></td>
<td>Worker3</td>
<td>76 77</td>
</tr>
</tbody>
</table>
Worker Fail Over

- Shard-0 Worker1
- Shard-1 Worker2
- Shard-2 Worker3

LeaseKey | LeaseOwner | LeaseCounter
---|---|---
Shard-0 | Worker1 | 85 86 87
Shard-1 | Worker2 | 94
Shard-2 | Worker3 | 76 77 78
Worker Fail Over

<table>
<thead>
<tr>
<th>LeaseKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-0</td>
<td>Worker1</td>
<td>85 86 87 88</td>
</tr>
<tr>
<td>Shard-1</td>
<td>Worker3</td>
<td>94 95</td>
</tr>
<tr>
<td>Shard-2</td>
<td>Worker3</td>
<td>76 77 78 79</td>
</tr>
</tbody>
</table>
Worker Load Balancing

<table>
<thead>
<tr>
<th>LeaseKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-0</td>
<td>Worker1</td>
<td>88</td>
</tr>
<tr>
<td>Shard-1</td>
<td>Worker3</td>
<td>96</td>
</tr>
<tr>
<td>Shard-2</td>
<td>Worker3</td>
<td>78</td>
</tr>
</tbody>
</table>
Worker Load Balancing

<table>
<thead>
<tr>
<th>LeaseKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-0</td>
<td>Worker1</td>
<td>88</td>
</tr>
<tr>
<td>Shard-1</td>
<td>Worker3</td>
<td>96</td>
</tr>
<tr>
<td>Shard-2</td>
<td>Worker4</td>
<td>79</td>
</tr>
</tbody>
</table>
Resharding

- Shard-0
- Shard-1
- Shard-2

Worker1

<table>
<thead>
<tr>
<th>ShardKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
<th>checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-0</td>
<td>Worker1</td>
<td>90</td>
<td>SHARD_END</td>
</tr>
</tbody>
</table>

Worker2
## Resharding

<table>
<thead>
<tr>
<th>Shard</th>
<th>Worker</th>
<th>LeaseKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
<th>checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-0</td>
<td>Worker1</td>
<td>Shard-0</td>
<td>Worker1</td>
<td>90</td>
<td>SHARD_END</td>
</tr>
<tr>
<td>Shard-1</td>
<td>Worker1</td>
<td>Shard-1</td>
<td>Worker1</td>
<td>0</td>
<td>TRIM_HORIZON</td>
</tr>
<tr>
<td>Shard-2</td>
<td>Worker1</td>
<td>Shard-2</td>
<td>Worker1</td>
<td>0</td>
<td>TRIM_HORIZON</td>
</tr>
</tbody>
</table>
Resharding

<table>
<thead>
<tr>
<th>Shard-0</th>
<th>Worker1</th>
<th>LeaseKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
<th>checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-1</td>
<td>Worker1</td>
<td>LeaseKey</td>
<td>LeaseOwner</td>
<td>LeaseCounter</td>
<td>checkpoint</td>
</tr>
<tr>
<td>Shard-2</td>
<td>Worker2</td>
<td></td>
<td></td>
<td></td>
<td>checkpoint</td>
</tr>
<tr>
<td>Shard-0</td>
<td>Worker1</td>
<td>LeaseKey</td>
<td>LeaseOwner</td>
<td>LeaseCounter</td>
<td>SHARD_END</td>
</tr>
<tr>
<td>Shard-1</td>
<td>Worker1</td>
<td>LeaseKey</td>
<td>LeaseOwner</td>
<td>LeaseCounter</td>
<td>TRIM_HORIZON</td>
</tr>
<tr>
<td>Shard-2</td>
<td>Worker2</td>
<td>LeaseKey</td>
<td>LeaseOwner</td>
<td>LeaseCounter</td>
<td>TRIM_HORIZON</td>
</tr>
</tbody>
</table>
Resharding

Shard-0
  Shard-1
  Shard-2

Worker1

Worker2

<table>
<thead>
<tr>
<th>LeaseKey</th>
<th>LeaseOwner</th>
<th>LeaseCounter</th>
<th>checkpoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shard-1</td>
<td>Worker1</td>
<td>2</td>
<td>TRIM_HORIZON</td>
</tr>
<tr>
<td>Shard-2</td>
<td>Worker2</td>
<td>3</td>
<td>TRIM_HORIZON</td>
</tr>
</tbody>
</table>
Putting this into production

Cost & Scale

500MM tweets/day = ~ 5,800 tweets/sec

2k/tweet is ~12MB/sec (~1TB/day)

$0.015/hour per shard, $0.014/million PUTS

Kinesis cost is $0.47/hour

Redshift cost is $0.850/hour (for a 2TB node)

Total: $1.32/hour
Design Challenge(s)

- Dynamic Resharding & Scale Out
- Enforcing Quotas (think proxy fleet with 1Ks servers)
- Distributed Denial of Service Attack (unintentional)
- Dynamic Load Balancing on Storage Servers
- Heterogeneous Workloads (tip of stream vs 7 day)
- Optimizing Fleet Utilization (proxy, control, data planes)
- Avoid Scaling Cliffs
- ...
Sushiro: Kaiten Sushi Restaurants

380 stores stream data from sushi plate sensors and stream to Kinesis
Sushiro: Kaiten Sushi Restaurants

380 stores stream data from sushi plate sensors and stream to Kinesis
Real-Time Streaming Data with Kinesis Streams

1 TB+/day game data analyzed in real-time | Gaming

5 billion events/wk from connected devices | IoT

17 PB of game data per season | Entertainment

100 billion ad impressions/day, 30 ms response time | Ad Tech

100 GB/day click streams
250+ sites | Enterprise

50 billion ad impressions/day
sub-50 ms responses | Ad Tech

17 million events/day
| Technology

1 billion transactions per day | Bitcoin
Streams provide a *foundational abstraction* on which to build higher level services.
Amazon Kinesis Firehose
Load massive volumes of streaming data into Amazon S3, Redshift and Elasticsearch

Zero administration: Capture and deliver streaming data into Amazon S3, Amazon Redshift, and other destinations without writing an application or managing infrastructure.

Direct-to-data store integration: Batch, compress, and encrypt streaming data for delivery into data destinations in as little as 60 secs using simple configurations.

Seamless elasticity: Seamlessly scales to match data throughput w/o intervention.
Amazon Kinesis Firehose

Fully Managed Service for Delivering Data Streams into AWS Destinations
Amazon Kinesis Analytics

Analyze data streams continuously with standard SQL

- **Apply SQL on streams:** Easily connect to a Kinesis Stream or Firehose Delivery Stream and apply ANSI standard SQL.
- **Build real-time applications:** Perform continual processing on streaming data with sub-second processing latencies.
- **Easy Scalability:** Elastically scales to match data throughput.

---

Connect to Kinesis streams, Firehose delivery streams

Run standard SQL queries against data streams

Kinesis Analytics can send processed data to analytics tools so you can create alerts and respond in real-time.
Realtime Analytics Patterns

• Simple counting (e.g. failure count)
• Counting with Windows (e.g. failure count every hour)
• Preprocessing: filtering, transformations (e.g. data cleanup)
• Alerts, thresholds (e.g. alarm on high temperature)
• Data Correlation, Detect missing events, detecting erroneous data (e.g. detecting failed sensors)
• Joining event streams (e.g. detect a hit on soccer ball)
• Merge with data in database, collect, update data conditionally
Realtime Analytics Patterns (contd.)

• Detecting Event Sequence Patterns (e.g. small transaction followed by large transaction)

• Tracking - follow some related entity’s state in space, time etc. (e.g. location of airline baggage, vehicle, customer by beacon)

• Detect trends – Rise, turn, fall, outliers, complex trends like triple bottom etc., (e.g. algorithmic trading, SLA, load balancing).
Amazon Kinesis: Streaming data made easy
Services make it easy to capture, deliver and process streams on AWS

**Kinesis Streams**
For Technical Developers
Build your own custom application to process or analyze streaming data

**Kinesis Analytics**
For all developers, analysts and data scientists
Easily analyze streaming data using standard SQL

**Kinesis Firehose**
For all developers, data scientists, IT professionals
Transform and load streaming data into S3, Redshift, Elasticsearch, and more...
Stream Processing End2End

Kinesis Analytics

**Analyze**
- Filter
- Temporal joins
- Combine w/ reference data
- Projections
- Correlate
- Windowed Aggregates
- Anomaly Detection
- ...

**Transform**
- Extract Fields
- Clean
- Enrich

Durable ingest, repeatable processing → In stream processing → low latency delivery to persist, alert, visualize
IoT Sensors (Example: Hello Inc.)

Sleep monitoring devices send data like bedroom temperature, humidity, ambient light, noise level, and particulate count.

Kinesis Streams reliably collects, stores, and exposes sensor data for processing.

DynamoDB enriches, aggregates, and transforms data for real-time per-user analyses.

Firehose loads data into S3 and Redshift for data science and durable storage.

Consumers get better sleep by monitoring and adjusting their sleeping conditions.
Customer Clickstream

- Send clickstream data to Kinesis Streams
- Kinesis Streams stores and exposes clickstream data for processing
- Custom application built on Kinesis Client Library makes real-time content recommendations
- Firehose loads data into S3 and Redshift for data science and durable storage
- Readers see personalized content suggestions and engage more
Closing Thoughts

Streaming data is highly prevalent and relevant; Stream data processing is on the rise; A key part of business critical workflows today, a powerful abstraction for building a new class of applications & data intensive services tomorrow. A rich area for distributed systems, programming model, IoT, and new service(s) research.
Questions