Real-time Analytics at Facebook
<table>
<thead>
<tr>
<th></th>
<th>Agenda</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Analytics and Real-time</td>
</tr>
<tr>
<td>2</td>
<td>Data Freeway</td>
</tr>
<tr>
<td>3</td>
<td>Puma</td>
</tr>
<tr>
<td>4</td>
<td>Future Works</td>
</tr>
</tbody>
</table>
Analytics and Real-time
what and why
Facebook Insights

- **Use cases**
  - Websites/Ads/Apps/Pages
  - Time series
  - Demographic break-downs
  - Unique counts/heavy hitters

- **Major challenges**
  - Scalability
  - Latency
Analytics based on Hadoop/Hive

- 3000-node Hadoop cluster
- Copier/Loader: Map-Reduce hides machine failures
- Pipeline Jobs: Hive allows SQL-like syntax
- Good scalability, but poor latency! 24 – 48 hours.
How to Get Lower Latency?

- Small-batch Processing
  - Run Map-reduce/Hive every hour, every 15 min, every 5 min, ...
  - How do we reduce per-batch overhead?

- Stream Processing
  - Aggregate the data as soon as it arrives
  - How to solve the reliability problem?
Decisions

- Stream Processing wins!

- Data Freeway
  - Scalable Data Stream Framework

- Puma
  - Reliable Stream Aggregation Engine
Data Freeway
scalable data stream
• Simple push/RPC-based logging system

• Open-sourced in 2008. 100 log categories at that time.

• Routing driven by static configuration.
Data Freeway

- 9GB/sec at peak, 10 sec latency, 2500 log categories
Calligraphus

- **RPC → File System**
  - Each log category is represented by 1 or more FS directories
  - Each directory is an ordered list of files

- **Bucketing support**
  - Application buckets are application-defined shards.
  - Infrastructure buckets allows log streams from x B/s to x GB/s

- **Performance**
  - Latency: Call sync every 7 seconds
  - Throughput: Easily saturate 1Gbit NIC
Continuous Copier

- File System → File System
- Low latency and smooth network usage

Deployment
-Implemented as long-running map-only job
- Can move to any simple job scheduler

Coordination
- Use lock files on HDFS for now
- Plan to move to Zookeeper
• File System $\rightarrow$ Stream ($\rightarrow$ RPC)

• Reliability
  - Checkpoints inserted into the data stream
  - Can roll back to tail from any data checkpoints
  - No data loss/duplicates
### Channel Comparison

<table>
<thead>
<tr>
<th></th>
<th>Push / RPC</th>
<th>Pull / FS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Latency</td>
<td>1-2 sec</td>
<td>10 sec</td>
</tr>
<tr>
<td>Loss/Dups</td>
<td>Few</td>
<td>None</td>
</tr>
<tr>
<td>Robustness</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td>Complexity</td>
<td>Low</td>
<td>High</td>
</tr>
</tbody>
</table>

- **Scribe**
- **Calligraphus**
- **PTail + ScribeSend**
- **Continuous Copier**
Puma
real-time aggregation/storage
Overview

- ~ 1M log lines per second, but light read
- Multiple Group-By operations per log line
- The first key in Group By is always time/date-related
- Complex aggregations: Unique user count, most frequent elements
## MySQL and HBase: one page

<table>
<thead>
<tr>
<th>Feature</th>
<th>MySQL</th>
<th>HBase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel</td>
<td>Manual sharding</td>
<td>Automatic load balancing</td>
</tr>
<tr>
<td>Fail-over</td>
<td>Manual master/slave switch</td>
<td>Automatic</td>
</tr>
<tr>
<td>Read efficiency</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Write efficiency</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Columnar support</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Puma2 Architecture

- PTail provide parallel data streams

- For each log line, Puma2 issue “increment” operations to HBase. Puma2 is symmetric (no sharding).

- HBase: single increment on multiple columns
Puma2: Pros and Cons

• Pros
  • Puma2 code is very simple.
  • Puma2 service is very easy to maintain.

• Cons
  • “Increment” operation is expensive.
  • Do not support complex aggregations.
  • Hacky implementation of “most frequent elements”.
  • Can cause small data duplicates.
Improvements in Puma2

- **Puma2**
  - Batching of requests. Didn’t work well because of long-tail distribution.

- **HBase**
  - “Increment” operation optimized by reducing locks.
  - HBase region/HDFS file locality; short-circuited read.
  - Reliability improvements under high load.

- Still not good enough!
Puma3 Architecture

- Puma3 is sharded by aggregation key.
- Each shard is a hashmap in memory.
- Each entry in hashmap is a pair of an aggregation key and a user-defined aggregation.
- HBase as persistent key-value storage.
Write workflow

- For each log line, extract the columns for key and value.
- Look up in the hashmap and call user-defined aggregation
• **Checkpoint workflow**
  
  - Every 5 min, save modified hashmap entries, PTail checkpoint to HBase
  
  - On startup (after node failure), load from HBase

  Get rid of items in memory once the time window has passed
Puma3 Architecture

- **Read workflow**
  - Read uncommitted: directly serve from the in-memory hashmap; load from Hbase on miss.
  - Read committed: read from HBase and serve.
Puma3 Architecture

- **Join**
  - Static join table in HBase.
  - Distributed hash lookup in user-defined function (udf).
  - Local cache improves the throughput of the udf a lot.
Puma2 / Puma3 comparison

- Puma3 is much better in write throughput
  - Use 25% of the boxes to handle the same load.
  - HBase is really good at write throughput.

- Puma3 needs a lot of memory
  - Use 60GB of memory per box for the hashmap
  - SSD can scale to 10x per box.
Puma3 Special Aggregations

- **Unique Counts Calculation**
  - Adaptive sampling
  - Bloom filter (in the plan)

- **Most frequent item (in the plan)**
  - Lossy counting
  - Probabilistic lossy counting
PQL – Puma Query Language

- CREATE INPUT TABLE t (‘time’, ‘adid’, ‘userid’);

- CREATE VIEW v AS
  SELECT *, udf.age(userid)
  FROM t
  WHERE udf.age(userid) > 21

- CREATE HBASE TABLE h …

- CREATE LOGICAL TABLE l …

- CREATE AGGREGATION ‘abc’
  INSERT INTO l (a, b, c)
  SELECT
    udf.hour(time),
    adid,
    age,
    count(1),
    udf.count_distinc(userid)
  FROM v
  GROUP BY
    udf.hour(time),
    adid,
    age;
Future Works
challenges and opportunities
Future Works

- **Scheduler Support**
  - Just need simple scheduling because the work load is continuous

- **Mass adoption**
  - Migrate most daily reporting queries from Hive

- **Open Source**
  - Biggest bottleneck: Java Thrift dependency
  - Will come one by one
Similar Systems

• STREAM from Stanford
• Flume from Cloudera
• S4 from Yahoo
• Rainbird/Storm from Twitter
• Kafka from Linkedin
Key differences

• **Scalable Data Streams**
  - 9 GB/sec with < 10 sec of latency
  - Both Push/RPC-based and Pull/File System-based
  - Components to support arbitrary combination of channels

• **Reliable Stream Aggregations**
  - Good support for Time-based Group By, Table-Stream Lookup Join
  - Query Language: Puma : Realtime-MR = Hive : MR
  - No support for sliding window, stream joins