The Unified Logging Infrastructure for Data Analytics at Twitter

Authors: George Lee, Jimmy Lin, Chuang Liu, Andrew Lorek, and Dmitriy Ryaboy

Presented by: Jason Florack
CIS 611 - Spring 2014
Sections

1. Abstract / Introduction
2. Scribe Infrastructure
3. Toward Unified Logging
4. Session Sequences
5. Applications
6. Ongoing Work
7. Conclusions
Abstract

- Large-scale data analytics has been a major focus in recent years
- But how is the data collected and structured to begin with?
- This paper presents Twitter’s production logging infrastructure, including an evolution from application-specific logging to a unified “client events” log format
Abstract (cont’d)

- Messages are captured in common, well-formatted, flexible Thrift messages
- **Session sequences** – compact summaries that can answer a large class of common queries quickly
Introduction

• Hadoop / MapReduce have made it easier for organizations to perform massive data analytics to better understand and serve customers and users

• The rise of social media and user-generated content, among other factors, has created new challenges and opportunities
Introduction (cont’d)

• Twitter has built a Hadoop-based platform for large-scale data analytics running Pig on a cluster of several hundred machines
• Twitter serves “traditional” business intelligence tasks such as cubing to support “roll up” and “drill down” of multi-dimensional data
• But these capabilities are not the subject of this paper
This paper’s focus is on how log data (~100TB/day) is structured in a consistent yet flexible format to support data analytics.

Another focus: how compressed digests called “session sequences” are materialized to speed up certain classes of queries.

Overall goal: streamline data analysis.
Twitter’s message delivery infrastructure is built on top of Scribe (pre-existing open source).

Events are identified using a consistent, hierarchical naming scheme, making it easy to identify data associated with a particular Twitter client, page, tab, etc.
Introduction (cont’d)

• This simplifies analyses, but logs are more verbose, which increases overall log volume and lowers query performance
• When logs arrive, they aren’t grouped into the most common unit of analysis: the user session
• So analysis must begin with scan of (lots of) data, then costly “group-by”
Session sequences

To support this common operation, summaries of client events are pre-materialized and organized by user session.

These summaries are 50 times smaller than the original logs, but still support large classes of common queries.
Goal of paper: share experiences in developing Twitter’s unified logging infrastructure

- Discussion of evolution from application-specific logging to unified logging format
- Description of a technique used to pre-materialize summaries of user sessions
Scribe Infrastructure

• System for aggregating high volumes of streaming log data in a robust, fault-tolerant, distributed manner
• Originated at Facebook in 2007
  • Still an integral part of Facebook’s logging infrastructure
• Open sourced in 2008
• Similar to more recent systems like Apache Flume and LinkedIn’s Kafka
Scribe (cont’d)

Twitter’s Scribe infrastructure
Scribe (cont’d)

- Scribe daemons on production hosts find aggregator hostnames via ZooKeeper, and send log messages to these aggregators
  - ZooKeeper provides abstraction of set of data nodes (znodes) organized like a file system
- Each log entry consists of two strings
  - Category
  - Message
Aggregators then deposit aggregated (e.g., per category) log data onto HDFS of the staging Hadoop clusters.

Periodic processes then copy data from these clusters into Twitter’s main Hadoop data warehouse.

Once everything is done, an hour’s worth of logs is slid into the main data warehouse.
Scribe (cont’d)

- At the end of the log moving process, logs are deposited in the main data warehouse in per-category, per-hour directories
  - e.g., /logs/category/YYYY/MM/DD/HH/
- Within each directory, log messages are bundled in a small number of large files, and are (in practice) partially time-ordered
Toward Unified Logging

**ANALYTICS PLATFORM:**

- Protocol Buffers and Thrift are two language-neutral data interchange formats that provide compact encoding of structured data
  - Both support nested structures; extensible
  - For logging, Thrift is preferred because it is also used widely in Twitter as a language-agnostic RPC framework
Unified Logging (cont’d)

• **Elephant Bird** –
  - Twitter’s system that specifies a data schema for a Hadoop job, from which the serialization compiler generates code to read, write, and manipulate data
  - Automatically generates Hadoop record readers and writers
Unified Logging (cont’d)

• **Pig** – a high-level dataflow language
  • Compiles into physical plans that are executed on Hadoop
  • Via Pig Latin, Pig provides concise primitives for expressing common operations such as projection, selection, group, join, etc.
  • Often used at Twitter for actual production jobs and ad hoc queries
Unified Logging (cont’d)

- **Oink** – Twitter’s workflow manager
  - Coordinates production analytics jobs by scheduling recurring jobs at fixed intervals
  - Handles dependencies between jobs
    - *Example*: if Job B requires data generated by Job A, Oink will schedule A, verify A has completed successfully, then schedule B
  - Preserves execution traces for auditing
    - When a job began, how long it lasted, etc.
- Oink schedules → Pig scripts → Hadoop jobs
Unified Logging (cont’d)

• HISTORY / MOTIVATION:
  • Initially, all applications defined their own, custom structure that they logged via Scribe
    • = “Application-specific logging”
  • However fast and flexible this approach was, it came with drawbacks when it came to using and managing the data
Unified Logging (cont’d)

• Drawbacks:
  • No consistency in message formatting (e.g., timestamps, naming conventions) makes it difficult to understand messages and/or configure even common tasks
  • Scribe categories are numerous and have potentially misleading names, causing a resource discovery problem
  • Making sense of joined data from multiple sources was a huge challenge
  • Pig analysis scripts were slow and error-prone due to varying data sources/formats
Unified Logging (cont’d)

- THE “CLIENT EVENTS” UNIFIED LOGGING FORMAT:

- A relatively recent effort within Twitter to develop a unified logging framework that simplifies analysis without imposing substantial additional burden on application developers
Unified Logging (cont’d)

- A hierarchical six-level naming scheme was imposed for all events

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>client</td>
<td>client application</td>
<td>web, iphone, android</td>
</tr>
<tr>
<td>page</td>
<td>page or functional grouping</td>
<td>home, profile, who_to_follow</td>
</tr>
<tr>
<td>section</td>
<td>tab or stream on a page</td>
<td>home, mentions, retweets, searches, suggestions</td>
</tr>
<tr>
<td>component</td>
<td>component, object, or objects</td>
<td>search_box, tweet</td>
</tr>
<tr>
<td>element</td>
<td>UI element within the component</td>
<td>button, avatar</td>
</tr>
<tr>
<td>action</td>
<td>actual user or application action</td>
<td>impression, click, hover</td>
</tr>
</tbody>
</table>

Table 1: Hierarchical decomposition of client event names.

- Six levels align with view hierarchy of Twitter clients
Unified Logging (cont’d)

• Example event:
  `web:home:mentions:stream:avatar:profile_click`

• Reading the event name from right to left, we can see that there was an image profile click on the avatar of a tweet in the mentions timeline for a user on twitter.com

• Hierarchical namespace makes it easy to slice-and-dice categories of events with simple regular expressions
  - `web:home:mentions:*` or `*:profile_click`
Unified Logging (cont’d)

- Also, Oink jobs automatically aggregate counts of events according to a number of schemas, which are then presented as top-level metrics, providing rudimentary statistics daily without any additional burden on the application developer.
Unified Logging (cont’d)

• Client event – a Thrift structure containing these components:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_initiator</td>
<td>{client, server} × {user, app}</td>
</tr>
<tr>
<td>event_name</td>
<td>event name</td>
</tr>
<tr>
<td>user_id</td>
<td>user id</td>
</tr>
<tr>
<td>session_id</td>
<td>session id</td>
</tr>
<tr>
<td>ip</td>
<td>user’s IP address</td>
</tr>
<tr>
<td>timestamp</td>
<td>timestamp</td>
</tr>
<tr>
<td>event_details</td>
<td>event details</td>
</tr>
</tbody>
</table>

Table 2: Definition of a client event.
Unified Logging (cont’d)

- `event_initiator` – specifies whether the event as triggered on the client side or the server side, and whether the event was user initiated or application initiated
- **All** client events contain fields for `user_id`, `session_id`, and `ip`
  - Explicit attempt to tackle previous issues with joining data from disparate sources
  - “Group-by “ suffices to reconstruct user sessions
Unified Logging (cont’d)

- Logging format is imposed across all Twitter clients: twitter.com, iPhone, iPad, Android phones, etc.
- Events of the same type across different clients are given the same name if possible
- Made possible by consistent design language across all Twitter clients
  - e.g., the Twitter terms “mentions” and “impression” mean the same thing everywhere
In summary, **client events** form the foundation of Twitter’s unified logging infrastructure, in two senses of the word “unified”:

- All log messages share a common format with clear semantics,
- Log messages are stored in a single place (as opposed to different Scribe category silos with application-specific logging)
Unified Logging (cont’d)

- Advantages compared to application-specific logging:
  - Consistent, hierarchical event namespace across all Twitter clients and a common message format makes logs easier to understand and more approachable
  - Common semantics for different fields in the Thrift messages makes it easier to reconstruct user session activity
  - A single location for all client event messages simplifies the resource discovery issue (i.e., knowing where to find what logs)
Session Sequences

- As mentioned previously, although the unified logging format has definite benefits, the logs tend to be more verbose, even after compression.
- So although analyzing user sessions became simpler, reconstructing sessions remained processing intensive – essentially, a large group-by across huge amounts of data.
Session Sequences (cont’d)

• Twitter discovered that large classes of queries could be answered using only the sequence of client event names within a user session, since the hierarchical namespace provides a lot of information on its own.

• Critically, these queries did not require going deeply into the details of Thrift messages.
Session Sequences (cont’d)

• Examples of this type of query:
  • Queries involving calculating Click-through rate (CTR) or Follow-through rate (FTR)
    • Example: How often are search results, who-to-follow suggestions, trends, etc. clicked on within a session, with respect to the number of impressions recorded?
    • With this type of case, details are less important; it suffices to know that an impression was followed by a “click” or “follow” event
  • Navigation behavior analysis queries
    • Example: How often do users take advantage of the “discovery” features?
    • These queries focus on how users navigate within Twitter clients; again, event names alone are sufficient
Session Sequences (cont’d)

- Since most of Twitter’s Pig scripts begin by reconstructing user sessions, it made sense to pre-materialize the sessions.
- However, only compact, compressed summaries of the sequence of client event names is produced.

= Session sequences
Session Sequences (cont’d)

- **Session sequence** – a sequence of symbols $S = \{s_0, s_1, s_2...s_n\}$ where each symbol represents a client event, and the sequence of symbols corresponds to the sequence of client events that comprise the user session

- Each symbol is represented by a unicode code point, such that any session sequence is a valid unicode string (sequence of unicode characters)
Session Sequences (cont’d)

• Mapping between events and unicode code points (i.e., the dictionary) is such that more frequent events are assigned smaller code points.

• Since each session sequence is a valid unicode string, analyses can be expressed in terms of regular expressions and other common string manipulations.
Session Sequences (cont’d)

- Construction of session sequences:
  1. A day’s logs are imported into the main data warehouse, and Oink triggers a job that scans the client event logs to look at event counts. The counts, along with samples of each type, are stored in HDFS.
  2. Sessions are reconstructed from the raw client events logs, via a group-by on user_id and session_id (delimited by 30 minutes of inactivity).
Session Sequences (cont’d)

- A session sequence is simply a unicode string that captures the names of the client events that comprise the session in a compact manner.
- User relation materialized on HDFS:
  ```
  user_id: long, session_id: string, ip: string, session_sequence: string, duration: int
  ```
- Note that only overall session duration is recorded; temporal information re: individual events is not available.
- Design choice: compact encoding over more granular data.
Session Sequences (cont’d)

• In summary, queries over session sequences are substantially faster than queries over the raw client event logs, both in terms of lower latency and higher throughput.

• The cost of increased performance = a restriction of the querying that can be done, down to a certain class of queries.
As part of an overall effort to enhance accessibility of their client event logs, Twitter maintains concise, up-to-date documentation.

Twitter has written an automatically-generated client event catalog and browsing interface which is coupled to the daily job of building the client event dictionary.

- Allows users to browse and search through the client events in a variety of ways: hierarchically, by each of the namespace components, and using regular expressions.
- Rebuilt every day → always up to date.
Applications

• **Summary Statistics**: a series of daily jobs generate summary statistics which feed into Twitter’s analytical dashboard called **BirdBrain**

• **Event Counting**: session sequences greatly speed up simple queries involving counting of events
Applications (cont’d)

• **Funnel Analytics**: originally developed in the context of e-commerce sites, these analyses focus on user attention in a multi-step process
  • **Example**: making an online purchase via a series of steps (reviewing shopping cart, entering shipping info, select shipping options, etc.)
  • Captures abandonment at each step; e.g., a user enters a shipping option but not payment details
  • In general, a site wants each user to flow through the entire funnel
  • Factors that impact completion rate: number, nature, layout, and design of each step of the process
Example in the context of Twitter:

Signup flow (sequence of steps taken by a user to join Twitter)

- Must be easy for new users but difficult for spammers attempting to create fraudulent accounts
- Must show novice users how to use the service without being boring

After evaluation of an arbitrary number of stages that make up a funnel in terms of client event names, the output might look something like:

(0, 490123) // 490123 people entered process
(1, 297071) // count that completed first stage etc.
Applications (cont’d)

• **User Modeling** – analytics used to identify “interesting” user behavior

• Techniques are borrowed from natural language processing (NLP)
  
  • **Language modeling**: as is the case with language models, how a Twitter user behaves right now is strongly influenced by immediately preceding actions; less so by an action 5 steps ago, and even less by an action 15 steps ago

  • **Collocations (commonly occurring patterns of words)**: A simple example is “hot dog” in language; in Twitter, it is possible to extract “activity collocates” → potentially interesting patterns
Ongoing Work

- Client events unified logging infrastructure went into production in Spring 2011
- Session sequences went into production Winter 2011
- At time paper was written (2012), a number of extensions were being worked on:
  1. Elephant Twin – a generic indexing infrastructure for handling highly-selective queries; used to complement session sequences in order to provide more detail than event names alone
  2. Drawing more from the field of natural language to come up with more advanced techniques
Conclusions

• “Data is the lifeblood of many organizations today, but it’s sometimes easy to forget that accumulating vast data warehouses is pointless in and of itself. What’s important are the insights that the data reveal about users and customers – and obtaining these insights in a timely fashion.”

• Supporting data analysis activities involves several different components:
  • Robust message delivery system
  • Flexibility at the point where log messages are generated
  • Each with which those messages can be manipulated downstream to perform data analytics
Conclusions (cont’d)

• Twitter uses Scribe and Thrift as their message delivery mechanism; these are mature and widely deployed
• For the other two components, at first Twitter used an application-specific architecture that made logging easy but analysis difficult
• Now, Twitter believes the unified client events logging infrastructure strikes a better balance between the desires of application developers (flexible logging) and the needs to data scientists (easy data analysis)
Conclusions (cont’d)

• Session sequences make common queries even easier and faster
• Twitter’s goal is to streamline the end-to-end analytics lifecycle, from messages logged at production servers to insights delivered by data scientists
Questions?

Thank you for your time.