Using Visualizations to Monitor Changes and Harvest Insights from a Global-Scale Logging Infrastructure at Twitter

Krist Wongsuphasawat and Jimmy Lin

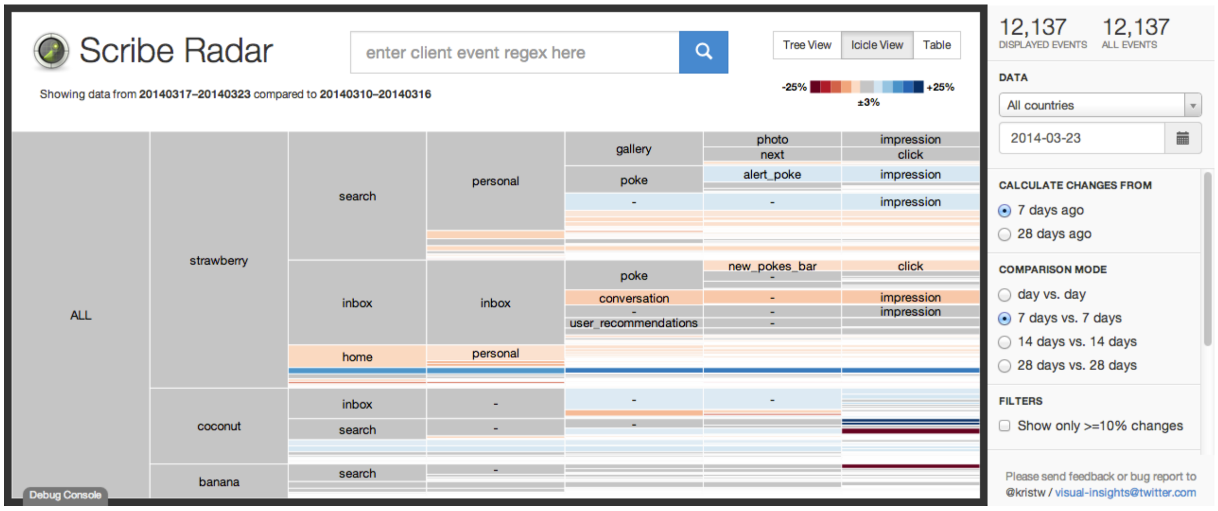


Fig. 1. Scribe Radar (Icicle View) — This interactive visualization displays a collection of log events from a hypothetical product that operates on three platforms: strawberry, coconut, and banana. Log events follow a six-level naming hierarchy (client:page:section:component:element:action). Here, we see that the event `strawberry:search:personal:-:-:impression` increased in frequency compared to seven days ago, indicated by a light blue rectangle, while a light red rectangle shows that the event `strawberry:inbox:inbox:conversation:-:impression` dropped slightly.

Abstract— Logging user activities is essential to data analysis for internet products and services. Twitter has built a unified logging infrastructure that captures user activities across all clients it owns, making it one of the largest datasets in the organization. This paper describes challenges and opportunities in applying information visualization to log analysis at this massive scale, and shows how various visualization techniques can be adapted to help data scientists extract insights. In particular, we focus on two scenarios: (1) monitoring and exploring a large collection of log events, and (2) performing visual funnel analysis on log data with tens of thousands of event types. Two interactive visualizations were developed for these purposes: we discuss design choices and the implementation of these systems, along with case studies of how they are being used in day-to-day operations at Twitter.

Index Terms—Information Visualization, Visual Analytics, Log Analysis, Log Visualization, Session Analysis, Funnel Analysis

1 Introduction

Many consumer internet companies depend on establishing a virtuous cycle that involves building a high-quality product, growing an engaged user base, and then learning from user activities to improve the product. Understanding user behavior based on activity logs forms an essential component of what we call *data science* today. These logs can be used for understanding how a product is used, determining which variant of a feature is better (A/B testing), and generating insights about user behavior. The advent of scalable, distributed, and fault-tolerant frameworks for data processing—especially the Hadoop open-source implementation of MapReduce [7]—has made it easier for organizations to perform "big data" analytics.

Previously, Twitter has built a *unified logging infrastructure* [23] that collects user activities across all clients it owns: all page loads, Tweets, logins, clicks, and other actions are logged and gathered in the organization's Hadoop data warehouses. It is the job of data scientists to clean, analyze, and transform these data into actionable *insights* that help software engineers refine their algorithms, product managers develop new features, and executives make more informed decisions.

However, extracting insights from log data is often a slow, difficult, and error-prone process, especially at the scale of petabytes. We see opportunities where information visualization can help. In particular, this paper focuses on two scenarios: (1) monitoring and exploring a large collection of log events, and (2) funnel analysis.

This work aims to bridge information visualization and large-scale data analytics. We emphasize that our contributions do not lie in the novelty of the visualization techniques, but rather in the integration of visualizations with a large-scale logging system. This paper makes the following contributions by detailing:

- Design choices, case studies, and lessons learned in applying information visualization to large-scale log analysis.
- An interactive visualization for exploring and monitoring a large collection of log events.
- An interactive visualization and a systematic approach to funnel analysis based on log events with customizable granularity.

- *Krist Wongsuphasawat is with Twitter, Inc.. E-mail: kristw@twitter.com.*
- *Jimmy Lin is with Twitter, Inc.. E-mail: jimmy@twitter.com.*

IEEE Symposium on Visual Analytics Science and Technology 2014
November 9-14, Paris, France
978-1-4799-6227-3/14/$31.00 ©2014 IEEE

Component	Description	Example
client	client application	`web`, `iphone`, `android`
page	page or functional grouping	`home`, `profile`, `who_to_follow`
section	tab or stream on a page	`home`, `mentions`, `retweets`, `searches`
component	component, object, or objects	`search_box`, `tweet`
element	UI element within the component	`button`, `avatar`
action	actual user or application action	`impression`, `click`, `hover`

Field	Description
`event_name`	event name
`user_id`	user id
`ip`	user's IP address
`timestamp`	timestamp
`event_details`	event details

Table 1. Hierarchical decomposition of client event names (left) and simplified definition of a client event (right).

The rest of this paper is organized as follows: We begin with an overview of data analytics at Twitter. Section 3 summarizes related work. Section 4 describes our application for monitoring events and Section 5 focuses on funnel analysis. We conclude with future plans in Section 6.

2 PROBLEM DOMAIN

Twitter provides a global platform for public self-expression and conversation in real-time across multiple clients: Twitter for iPhone, Twitter for Android, the Twitter website, TweetDeck, etc. The company's data warehouses provide an analytics platform that allows data scientists and engineers to better understand user activities across a multitude of usage scenarios and build useful data products. The analytics platform is comprised of thousands of Hadoop nodes across multiple datacenters, ingesting over one hundred terabytes of raw data every day. Engineers and data scientists from dozens of teams run tens of thousands of Hadoop jobs collectively; these jobs accomplish everything from data cleaning to simple aggregations and report generation to training machine-learned models for promoted products, spam detection, follower recommendation, and much more.

2.1 Logging at Twitter

2.1.1 Scribe

Twitter uses Scribe [36], a system for aggregating high volumes of streaming log data in a robust, fault-tolerant, and distributed manner; see details in [23]. A Scribe daemon runs on every production host and is responsible for sending local log data across the network to a cluster of dedicated aggregators in the same datacenter. Each log entry consists of two strings, a category and a message. The category is associated with configuration metadata that determines, among other things, where the data are written.

The aggregators in each datacenter are co-located with a staging Hadoop cluster. Their task is to merge per-category streams from all the server daemons and write the merged results to the Hadoop Distributed File System (HDFS) of the staging Hadoop cluster, compressing data on the fly. Another process is responsible for moving these logs from the per-datacenter staging clusters into the main Hadoop data warehouse. It applies certain sanity checks and transformations, such as merging many small files into a few big ones and building any necessary indexes. At the end of the log mover pipeline, data arrive in the main Hadoop data warehouse and are deposited in per-category, per-hour directories on HDFS (e.g., `/logs/category/YYYY/MM/DD/HH/`).

Each application writes logs using its own Scribe category. In practice, this creates a resource discovery problem, in that there are dozens of Scribe categories, many non-intuitively named or whose contents have substantially diverged from when the Scribe category was first established, making the category name meaningless at best or misleading at worst. Since the application developers are often disjoint from the data scientists who perform the analyses downstream, it is sometimes difficult, particularly for new data scientists, to even figure out what logs are available. As documentation falls out of sync with code over time, we often rely on tacit knowledge passed along through group mailing lists and by word of mouth.

2.1.2 Client Events

To address the challenges described above, "client events" represent an effort within Twitter to develop a unified logging framework to simplify analyses without imposing substantial additional burden on application developers.

Generalizing the notion of Scribe categories, the core idea is to impose a hierarchical six-level naming scheme for *all* events (comprised of client, page, section, component, element, action), outlined in Table 1 (left). This six-level decomposition aligns with the view hierarchy of Twitter clients. For example, in the case of the main web client (i.e., the twitter.com site), the namespace corresponds to the page's DOM structure, making it possible to automatically generate event names and thereby enforce consistent naming. This also enables reverse mappings; that is, given the event name, we can determine the DOM element where the event was triggered.

For example, `web:home:mentions:stream:avatar:profile_click` is triggered whenever there is an image profile click on the avatar of a Tweet in the mentions timeline for a user on twitter.com ("reading" the event name from right to left). This hierarchical namespace makes it easy to slice-and-dice categories of events with simple regular expressions to focus on an *ad hoc* grouping of interest. For example, analyses could be conducted on all actions on the user's home mentions timeline on twitter.com by considering `web:home:mentions:*`; or track profile clicks across all clients (twitter.com, iPhone, Android, etc.) with `*:profile_click`.

A client event itself is a structure that contains the components shown in Table 1 (right); the structure is simplified, but still preserves the essence of the design. The `event_details` field holds event-specific details as key-value pairs. For example, in the profile click event described above, the details field would hold the id of the profile clicked on. For an event corresponding to a search result, the `event_details` field would hold the target URL, rank in the results list, and other such information. Since different teams can populate these key-value pairs as they see fit, the message structure can be flexibly extended without central coordination.

In summary, client events form the foundation of our unified logging infrastructure in two senses of the word "unified": first, in that all log messages share a common format with clear semantics, and second, in that log messages are stored in a single place (as opposed to different Scribe category silos with application-specific logging).

2.1.3 Session sequences

User sessions, delimited by a 30-minute inactivity interval, form the starting point of many analytics queries. Many Hadoop jobs begin by reconstructing these user sessions from client events. Therefore, it makes sense to simply pre-materialize the sessions. Because the hierarchical event namespace provides a lot of information alone and many queries can be answered using only the sequence of client event *names* within a user session, we omit the `event_details` and encode only the event names in compressed summaries called *session sequences*.

A session sequence is defined as a sequence of symbols $S = \{s_0, s_1, s_2 ... s_n\}$ where each symbol is drawn from a finite alphabet Σ. We define a bijective mapping between Σ and the universe of event names, so that 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, i.e., sequence of unicode characters. Furthermore, we define the mapping between events and unicode code points (i.e., the dictionary) such that more frequent events are assigned smaller code points. This in essence captures a form of variable-length coding, as smaller code points require fewer bytes to physically represent. For example, the event `web:home:mentions:stream:avatar:profile_click` might get mapped to `\u0235`. Unicode comprises 1.1 million available code points, and it is unlikely that the cardinality of our alphabet will exceed this. Since each session sequence is a valid unicode string, we can express analyses in terms of regular expressions and other common string manipulations. Note that these session sequences are not meant for direct human consumption, and they need to be decoded first to be human-readable.

2.2 Data Analysis at Twitter

Academic researchers often focus on tasks that have well-defined goals, clean data, and clear success metrics. However, problems that data scientists in industrial settings face on a daily basis are far more "messy". Let us illustrate with a realistic but hypothetical scenario: The analysis typically begins with a poorly formulated problem, often driven from outside engineering and aligned with strategic objectives of the organization, e.g., "we need to attract a specific group of users". It is the data scientist's job to operationalize vague directives into a concrete, solvable problem, with the prospect that the solution will lead to a better understanding of how to fulfill the objective (attract users). For example:

- When does this group of users typically log in and out?
- How frequently?
- What features of the product do they use?
- Do they behave differently from other groups of users?
- Do these activities correlate with engagement?

Data identification is the first step in answering these questions. Data scientists need to identify relevant data from the many data sources available (Tweets, the follower graph, client events, etc.). It is common for new data scientists to be overwhelmed by the volume and variety of data available. Even within a single data source, finding the right subset to look at is not an easy task.

Data identification is usually followed by *exploratory data analysis*, which invariably reveals data quality issues (formatting bugs, corrupted records, etc.) [17]. In our recollection, we have never encountered a large, real-world dataset that was directly usable without data cleaning. When data collection errors are discovered, the engineers who are responsible are notified to fix the issues. Unfortunately, this type of verification is often too late, and logging errors often go unnoticed until the data are required for analysis. Typically, data cleaning and verification is an iterative process that is intertwined with data exploration. These activities are often performed by running Hadoop jobs to retrieve and pre-process the data, followed by analysis in R, Tableau, or other tools.

After exploratory data analysis, the data scientist can more precisely formulate the problem, cast it into a data mining or machine learning task, and define metrics for success. With the task more clearly defined, standard techniques such as classification, regression, clustering, etc. can be brought to bear in a manner that most academic researchers would be familiar with. The data analysis lifecycle concludes with the delivery of an insight, an actionable plan, or a data product; the cycle then starts afresh with the next analysis.

2.3 Motivation for this Work

There is clearly much room for improvement in the data analysis lifecycle: we see opportunities to leverage information visualization to improve data identification and exploratory analysis. Our work focuses on log data, particularly client events, because it is one of the most important datasets at Twitter. Furthermore, log analysis is a sufficiently common and general problem that our lessons learned can be applied to other systems and large-scale analytics tasks.

The first scenario we examined is monitoring and exploring the collection of client events. There are multiple roles at Twitter whose responsibilities depend at least in part on these data. The first role is the data scientist: identifying the client events related to their analyses is challenging since all Twitter clients contribute to the unified logging infrastructure. Currently, there are tens of thousands of client event types, whose numbers are still growing. What are the client events related to "log in"? The second role is the software engineer, who implements the data collection mechanisms (typically as part of product features). We would like to provide more visibility into the event stream so they can more easily spot errors and improve overall data quality. There are also product managers and engineering managers who occasionally check the movements of client events as indicators of product performance. This scenario and our proposed solution, a system called *Scribe Radar*, are detailed in Section 4.

The second scenario is *funnel analysis*, a common analytics task that explores patterns of event sequences. By applying information visualization techniques for event sequences [44] to the session sequences, we aim to provide data scientists a *visual* interface for exploring user sessions—the hope is that interesting behavioral patterns will map into distinct visual patterns, such that insights will literally "leap off the screen". The volume of data forced us to rethink standard visualization techniques and we developed data transformations to pre-aggregate the data. The entire process and visualization system called *Flying Sessions* are explained in Section 5.

Again, we recognize that the visualizations in this paper are adaptations of existing techniques and do not claim novelty in this regard. Instead, the value of this paper lies in real-world experiences of how we applied information visualization techniques at a large scale. We also share case studies of how our colleagues were able to derive insight using our applications.

3 Related work

3.1 Tree Visualizations

Most of the data in this work are trees or raw data that are transformed into trees. The most common way to display a tree, or hierarchy, is a node-link layout in 2D [42, 31], 3D [32], or hyperbolic space [22, 26]. Space-filling techniques, such as treemaps [16], icicle trees [20, 9] and sunburst trees [34] use implicit containment and geometry features to present a hierarchy. Icicle trees, also called icicle plots, display hierarchical data as stacked rectangles, usually ordered from top to bottom. The root takes the entire width. Each child node is placed under its parent with the width proportional to the percentage it consumes relative to its siblings. This visualization directly inspired our design (Figure 8). Another variation orders the tree from left to right, placing the children to the right of instead of under their parent (Figure 1).

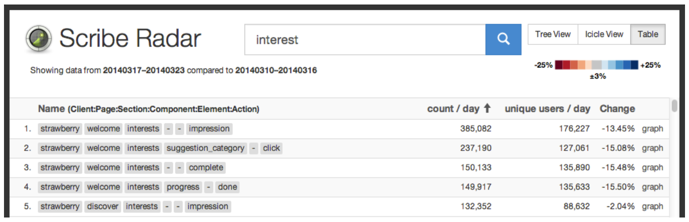
Another thread of work focuses on making comparisons, which can be categorized according to complexity factors of the differences [12]: T – topological changes (added/removed nodes, moved subtrees), V – node value changes, and I – whether the interior nodes' values are independent from their children's values. One group of work focuses on comparing only topological changes (T). Common techniques place the compared trees side-by-side and provide interactions or visual cues that highlight the differences [27]. Another group of work focuses on node value changes without topological changes (V), mostly involving treemaps [40, 37]. The Map of the Market [40] represents stock market price changes using color-coded treemaps. Recently, StemView [12] was developed to support all types of value changes and to provide partial support for topological changes, showing only added/removed nodes but not subtree movements ($t+V+I$). *Scribe Radar* (Section 4) falls into the same class ($t+V+I$). It compares two client event hierarchies to show partial topological changes (new/deleted events) and value changes (increased/decreased event counts). Although the interior nodes' values are aggregated values from their children in our scenario, the visualization can support cases when interior values are independent from their children as well. There are two main differences from StemView: First, our display shows a color-coded icicle tree ordered left-to-right instead of a mixture of bar charts and icicles, providing more intuitive interpretability. Second, interactions provided by the regular expression search box introduce new exploration capabilities.

3.2 Event Sequence Visualizations

Session sequences represent a type of *event sequence*, i.e., a sequence of timestamped events. A number of researchers have explored vi-

Date	Client	Page	Sec.	Comp.	Element	Action	Count
20131128	iphone	home	-	-	user	follow	80
20131128	iphone	home	-	-	menu	dismiss	2
20131129	iphone	home	-	-	user	follow	100
20131129	iphone	home	-	-	menu	dismiss	2

Table 2. Sample rows of client event counts in the database (simulated data); counts are generated on a daily basis.



Showing data from **20140317–20140323** compared to **20140310–20140316**

	Name (Client:Page:Section:Component:Element:Action)	count / day ↑	unique users / day	Change	
1.	strawberry welcome interests - - impression	385,082	176,227	-13.45%	graph
2.	strawberry welcome interests suggestion_category - click	237,190	127,061	-15.08%	graph
3.	strawberry welcome interests - - complete	150,133	135,890	-15.48%	graph
4.	strawberry welcome interests progress - done	149,917	135,633	-15.50%	graph
5.	strawberry discover interests - - impression	132,352	88,632	-2.04%	graph

Fig. 2. Scribe Radar (Table View) — A simple table that lists event names and lets users sort by different criteria.

sualization techniques for such data. Many early systems focused on visualizing a single record [2, 13, 18, 30]. The most common approach is to place events on a horizontal timeline according to the time that events occurred. Later, attention shifted towards visualizing multiple records in parallel. One popular technique is to stack instances of single-record visualizations and to provide additional functionality for aligning [39], searching [8, 38, 45], filtering [39], and grouping [4, 28]. However, as event sequence databases became larger, techniques that can provide abstractions of multiple event sequences are needed. *LifeFlow* [44] introduces a way to aggregate and provide an abstraction for multiple event sequences. LifeFlow's aggregation combines multiple event sequences into a tree based on an *alignment point*. *Outflow* [43] combines multiple event sequences into a graph of states based on the assumption that events are persistent and accumulate overtime. LifeFlow's successor *EventFlow* [25] extends support to events with intervals, e.g., ("lunch", 12-1pm) instead of ("lunch", 12pm). It handles overlaps and includes new simplification methods for intervals. Since client events do not have duration and the events are not persistent, we had to adapt LifeFlow to funnel analysis and also modify the technique to scale to hundreds of millions of records.

3.3 Visual Log Analysis

Visual log analysis can be traced back to user paths and web traffic visualizations [29, 15, 14, 6] in the time when websites were less dynamic and mostly consisted of a fixed set of HTML pages. Usability log visualization [11] is another group of work dedicated to supporting a small number of sessions from usability studies. There are also systems designed for computer log analysis, such as [35].

Later advances increased the amount and complexity of log data that could be visualized. Work in network security visualization, such as VAFLE [10], focus on analyzing raw packet captures, IDS alerts, firewall logs, etc. Our work is closer to another thread of work that analyzes logs to learn about user behavior. Session Viewer [21] was designed primarily to analyze web search logs, where users visit arbitrary pages across many domains. There is also increasing integration with large scale computing [33] and data mining techniques [5, 41]. One closely-related work both in terms of scale and technique is *Trail-Explorer2* [33], where the authors performed session analysis on top of MapReduce at eBay. However, there are a few differences: Trail-Explorer2 requires users to define a *sequence* of events or funnel (e.g., CheckOut → PaymentReview → PaymentConfirm → CheckoutSuccess) and then analyzes only the defined funnel (including time spent within each step). Our work (Section 5) asks for a *set* of events to be included and collects all possible funnels that can be generated from the given set of events. We also support dynamic event definition and let users define custom events. Finally, we provide additional data transformation to aggregate rare patterns.

4 Monitoring and Exploring Client Events

Standard summary statistics of all client events are computed daily using Hadoop and stored in Vertica, a column-oriented DBMS; see sample data in Table 2. Daily counts of each event type are displayed in a dashboarding system as a time series line chart, resulting in one line chart per each event type. Changes in these time series indicate usage trends, impacts after feature releases, logging errors, and other anomalies. However, it is impractical to inspect all these time series manually, and there is no easy way to obtain an overall sense of which events are increasing or decreasing in volume.

Furthermore, exploring or searching for event names using the existing interface was inconvenient. Users must select each level of the client event hierarchy until they locate the events that they are looking for: "client" and "page" are quite straightforward, but the "section", "component" and "element" levels are much more loosely defined. Many of these fields are empty. For instance, when looking for client events related to `tweet_composer`, one might wonder if it is a "section", "component" or "element"?

To address these issues, we built an interactive visualization to support monitoring and exploring client events.

4.1 Design

We gathered requirements through discussions with potential users (mainly engineers and data scientists) and defined the following goals:

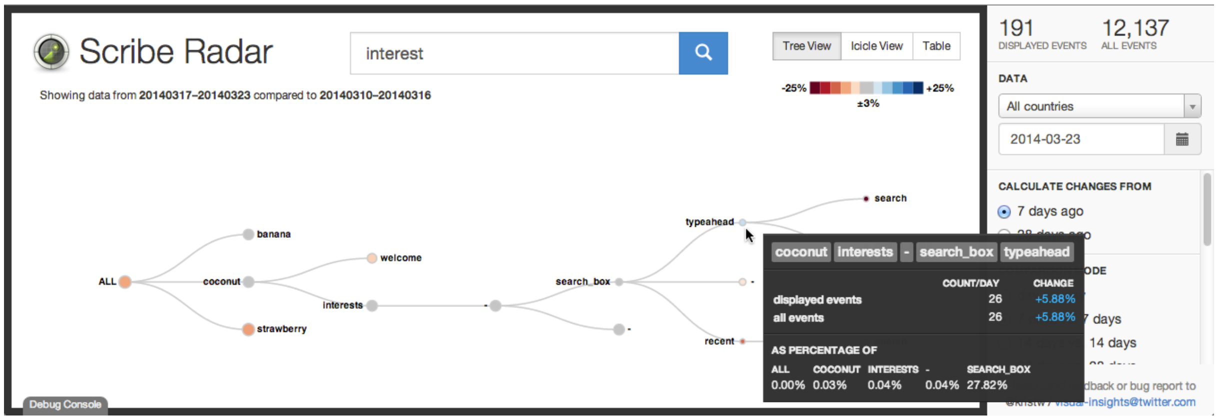
1. *Users can search for events.* The six-part event names are hard to remember, so the search feature should not require the user to know (or guess) the level of the hierarchy they are interested in. Typos or invalid event names should also be expected. The interface should not discourage users from retrying the search when getting wrong or empty results.
2. *Users can understand the event collection better.* Although users are aware that client events are designed using the six-level hierarchy, this can be very abstract since most have never seen all the events together, nor have they seen how each event contributes to the overall structure. We should make these hierarchical relationships more concrete and help users see the "big picture" more clearly. For example, what are all the events under `iphone:home:profile`? What is the proportion of events under `iphone:home:profile` relative to `iphone:home`?
3. *Users can notice changes in event volumes.* Users should be able to detect events that have changed in volume significantly in the past *n* days without clicking through thousands of graphs by hand.

4.2 Description of the system

Our visualization tool, called *Scribe Radar*, is shown in Figure 1. The front-end was implemented in Javascript using D3.js [3] and AngularJS, while the back-end was written in Ruby on Rails, providing access to data in Vertica. Scribe Radar provides multiple views of the client events collection: icicle tree, table, and node-link diagram, shown in Figures 1, 2 and 3, respectively. Users can search for events (top box) and click through to see details on demand (Figure 4). They can also customize the date range and filter events by properties such as the user's country. The data transformation flow is illustrated in Figure 5 and our design choices are explained as follows:

Implement search as a filter. To provide rapid interaction, search is implemented as a filter. A standard textbox serves as the main input: in the basic use case, users can perform simple keyword search. The textbox also accepts regular expressions to provide more control over results. When a search is submitted, the application retains only matching event types and displays filtered events without refreshing the page. Animated transitions transform previous results into new results, providing a seamless experience.

Use the event collection as the focus of attention. Whether users want to search for events or monitor changes, the output is always a collection of events, so we focus on visualizing such data and build a visual display that can show overviews and highlight changes.



Showing data from 20140317–20140323 compared to 20140310–20140316

coconut | interests | - | search_box | typeahead

	COUNT/DAY	CHANGE
displayed events	26	+5.88%
all events	26	+5.88%

AS PERCENTAGE OF

ALL	COCONUT	INTERESTS	-	SEARCH_BOX
0.00%	0.03%	0.04%	0.04%	27.82%

Fig. 3. Scribe Radar (Tree View) — A node-link diagram view of the event hierarchy. Here, the user is examining events related to `interest`.

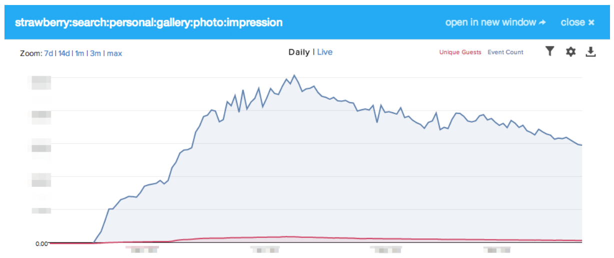

strawberry:search:personal:gallery:photo:impression
open in new window
close ×
Zoom: 7d | 14d | 1m | 3m | max
Daily | Live
Unique Guests
Event Count
0.00

Fig. 4. Scribe Radar (Time Series) — The details-on-demand pop-up that shows changes of each client event over time as a line chart.

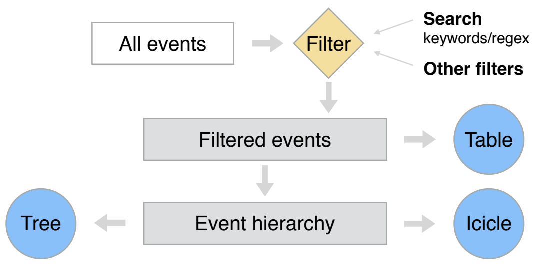


Fig. 5. Scribe Radar — Transformation from raw events to visualizations.

Build an event hierarchy based on the six-level namespace. As shown in Figure 6, an event hierarchy can be constructed from a collection of events. Rows of events and their counts on given dates are queried from Vertica. To calculate changes, we build two separate event hierarchies using data from different date ranges and then compute a diff tree, which is small enough to store in browser memory.

Provide multiple views of the event collection. We want to provide users with alternative views to support different tasks. The first view is a standard table of event names that users can sort by different columns. This view is simple, fast to render, and has almost no learning curve. However, it cannot provide an overview of the entire collection or show the structure of the events.

To display the event hierarchy, we considered a few alternative tree visualizations and evaluated prototypes informally with a few colleagues. A standard collapsible/expandable node-link tree was chosen first due to its simplicity and familiarity to users. However, with a large and bushy tree, expanding all nodes at once results in heavy occlusion and fails to highlight changes in the nodes.

To make better use of space, we considered a few space-filling techniques. The first prototype was implemented using a treemap [16]. However, the six-level structure became less explicit and identifying the levels of inner nodes can be challenging. Our next idea was to use a sunburst tree [34] because it displays each inner node in its own block rather than overlaying on the leaf nodes and separates nodes into one ring for each level, but the left side of the sunburst tree, where the levels are placed “backwards”, is unnatural for reading. Ultimately, the icicle tree was chosen due to its efficient use of space and readability. We laid out the icicles left to right, thus the height of each rectangle encodes the volume of events and its color represents changes. The red-to-blue sequential color palette encodes relative change for each client event from negative to positive. Darker colors indicate more dramatic changes. Users can click on each node to zoom into a subtree of the selected node. The icicle view has emerged as our preferred approach for visualizing the event collection.

We had also considered StemView [12], an icicle-based visualization that adds more visual encodings for displaying changes. However, the downside is that the visualization is less explicit in showing hierarchical structure and requires additional explanation. In our initial implementation, we wanted the visualizations to be self-explanatory with a low learning curve.

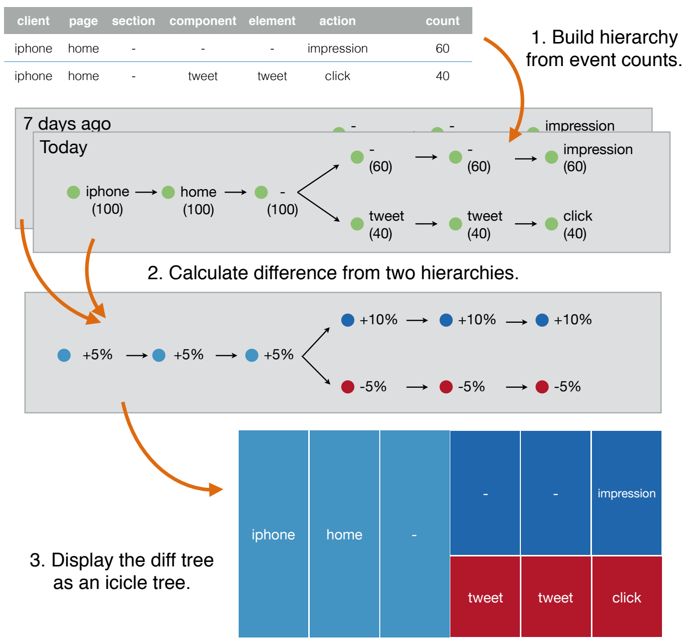
Show time series as details-on-demand. Scribe Radar follows the visual analytics mantra [19] “Analyze first, show the important, zoom, filter and analyze further, details on demand.” It analyzes the client event counts to compute the event hierarchy and diff tree; highlights important changes in the visualization; lets users zoom and filter by clicking or searching. The final component is to provide details on demand for the selected event. Users can click on any rectangle (in the icicle view) or any node (in the node-link diagram view) to open a pop-up dialog that displays the time series for the selected client event (Figure 4). In this pop-up, users can toggle between long-term data (aggregated daily) and real-time data (updated every minute). When users see rectangles with intense colors (indicating a significant change), they can click on it to inspect the changes.

4.3 User Feedback and Use Cases

Scribe Radar has been deployed inside Twitter since December 2013. According to the access logs, as of June 2014 (6 months after deployment), Scribe Radar has been visited more than 1,500 times and has been used by more than 500 unique users within the company; there are 8.72 unique visitors per day on average. To gather feedback, we sent a short questionnaire to an internal mailing list that includes engineers, data scientists, and product managers. We also received feedback from users via email and had discussions with them while providing technical support. Our findings are summarized as follows:

Learning and adoption. Unlike standard usability studies or controlled experiments, most users received no training from us and had to figure out how to use the tool by themselves. Many users learned about the tool from an announcement email that describes a few examples or heard about the tool from colleagues. Only a small number of users had seen a live demo.

Use cases. With broad adoption within the company among different groups of users (data scientists, engineers, and product managers), we can identify three main use cases:



client	page	section	component	element	action	count
iphone	home	-	-	-	impression	60
iphone	home	-	tweet	tweet	click	40

Fig. 6. Building the event hierarchy — This diagram explains how we construct an event hierarchy from two sets of events. (simulated data)

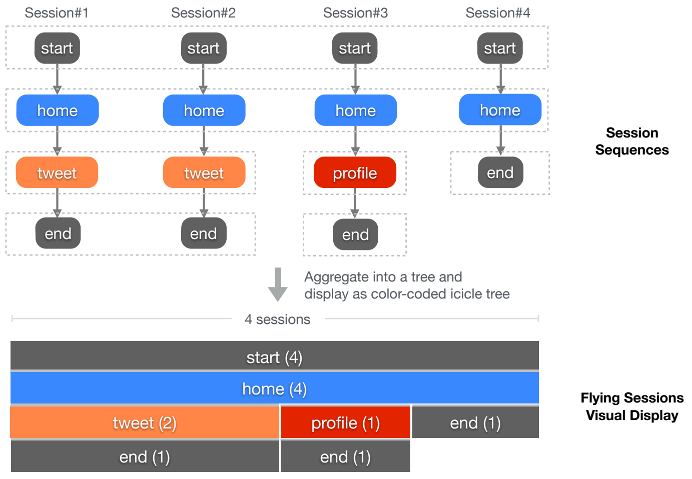


Fig. 7. Transforming session sequences into an overview visual display.

1. *Searching for client events of interest.* This clearly matches one of the design goals. When asking for users' feedback about the things that they liked the most, here are some of their answers: "great for looking up data for a particular set of events", "what I liked the most was the client [events] searching methods".
2. *Monitoring overall changes in client events.* Some users look for unexpected changes with Scribe Radar; this corresponds to the other main use case we designed for. One product manager told us about an incident where he noticed an unexpected intense red rectangle in the visualization. He was familiar with this client event and did not expect it to drop. He quickly suspected that the drop might be due to a logging issue. Investigating further, he entered a part of that client event name in the search box and found another event which was a typo of that event (e.g., *click* vs. *clcik*). Based on this, his team was able to identify the code change that introduced this typo and fixed the issue. This incident shows how the visualization helped to detect issues and can be used to improve overall data quality by providing better visibility.
3. *Examining behavior effects after product feature launches.* This mode of usage was discovered during development. By filtering the event collection, users can see only changes in events of interest. For example, after a product release that launched feature X on a particular date, we can determine how a user behavior of interest (e.g., marking a Tweet as a "favorite") was affected, compared to before the feature release. This analysis can be quickly performed by selecting the date and entering `favorite$` in the search box.[1] To narrow down to the iPhone only, for example, the user can refine the search to `iphone:.*:favorite$`.

5 Funnel Analysis Based on Log Events

Funnel analysis [1, 24], originally developed in the context of e-commerce sites, focuses on user attention in multi-step processes. A classic example is the purchase checkout flow, where one funnel might be: users visit their shopping carts, enter billing and shipping addresses, select the shipping option, enter payment details, and finally confirm. The funnel analogy is apt because it captures abandonment at each step, e.g., users never complete the purchase. An e-commerce site wants to maximize the number of users that flow through the entire funnel (i.e., complete a purchase) along with related metrics (e.g., total revenue). The number and nature of each step in the funnel plays an important role; for example, shortening the number of steps potentially comes at the cost of making each step more complex; on the other hand, users have little patience for activities that require too many steps. Companies typically run A/B tests to optimize the flow and to assess the impact on end-to-end metrics.

In the context of Twitter, there are a variety of complex funnels. An important one is the sign up flow, which is the sequence of steps taken by a user to join the service. For example, we might ask how many users went through the funnel (on a particular day): *front page* $\rightarrow$ *sign up page* $\rightarrow$ *sign up completed*?

Previously, we implemented simple Hadoop-based jobs [23] to count occurrences of specified funnels based on the session sequences described in Section 2. These jobs read a collection of session sequences and count the number of sessions with a given pattern, defined as an array of regular expressions of event names (simplified in the following pseudocode):

```
Q1 = countSequenceMatches(sequences, [
  'web:front:-:-:-:impression',
  'web:signup:form:-:-:impression',
  'web:signup:form:-:-:completed'
]);
```

The query Q_1 returns a count of the funnel *front page* $\rightarrow$ *sign up page* $\rightarrow$ *sign up completed*. However, this is only one possible path in the user interaction flow. Users might visit the sign up page but do not complete the sign up process. Another query Q_2 would be required to count all sequences of the form *front page* $\rightarrow$ *sign up page*. The number of visitors who did not complete the sign up flow can be computed by $Q_2 - Q_1$. For more complex analyses, there are multiple steps (e.g., import contact list) and actions (e.g., import, skip) that users can perform between the initial and terminal events. In order to learn at which step users "gave up", we need to include the corresponding event types and submit additional queries.

Instead of trying to count all the "funnel fragments" and combine them to understand the entire sign up flow, it would be more effective to provide an overview of all possible paths users take after they visit the sign up page. A data scientist should be able to simply ask "What happens after users visit the sign up page?" and analyze the results, without needing to explicitly specify all funnel combinations.

Based on the *LifeFlow* [44] technique for visualizing event sequences, multiple session sequences can be aggregated into an overview visualization. The original technique shows time gaps between events, but the session sequences do not explicitly encode timestamps (recall that by design they capture sequences of event names only). Thus, we decided to simplify the visual display and omit the time dimension, as illustrated in Figure 7. However, even without the time dimension, the LifeFlow technique has never before been applied to such a large number of event sequences and event types, and this

[1]The symbol `$` indicates "end of word" in regular expressions.

represented a challenge that we needed to overcome. Not surprisingly, an attempt to visualize the raw session sequences with the LifeFlow technique “out of the box” failed: there were too many patterns with too many event types, making any interpretation very difficult.

5.1 Design

The client events logging infrastructure was explicitly designed to record as much detail as possible. Thus, trying to visualize the raw session sequences introduces too many irrelevant distractions and is too slow. The expected data for the visualization are unique sequences and their counts. Ultimately, we need to reduce the number of unique sequences (Q) to reveal insights. There are two main factors that contribute to the number of unique sequences:

$$Q \propto T^L$$

1. *Number of event types (T):* Many raw sequences are unique or occur only a few times due to the large number of event types.
2. *Sequence length (L):* Entire session lengths could be anywhere from one to thousands of events (outliers). However, an analysis is likely to focus on only a small part of each session.

We collaborated closely with data scientists to identify common themes in questions they ask using session sequences and tried to map them to the two optimization goals above. We met a few times per week to receive feedback and iterate on prototypes. A few initial attempts to reduce the number of unique patterns included (1) limiting the sessions to only the first n events, (2) ignoring less frequent sequences, (3) focusing only on *client* and *page* from the six-level hierarchy, and a few more. Approach (1) did not work because the granularity of the log events is too fine. A simple action such as logging in can generate a multitude of events since it involves multiple back-end services that together render the home view. Thus, even for relatively large values of n, little useful information beyond the first few user actions are shown. Approach (2) ended up ignoring a significant portion of the sessions; in fact, rare sessions may be exactly what the data scientist was searching for. Approach (3) was promising and started to reveal interesting patterns. However, the fixed *client - page* abstraction limited the flexibility of the analysis and failed to show finer-grained interaction information. This approach was also unable to group together multiple pages that were less important in order to reduce the number of unique patterns.

Learning from initial prototypes, we developed a general template for the question that our funnel analysis tool was designed to answer: **What are the sequences of (set of events) (before/after) (alignment point)?** For example, what are the sequences of (pages) (after) (users look at the home timeline)? The data scientist must define *pages* and *users look at the home timeline* from raw client events, and from that specification our tool generates an interactive visualization that answers the question. Thus, the system supports these tasks:

1. *Map raw event types to custom event types.* For example, the user might define a mapping that collapses all Tweet events on the iPhone to an aggregate new event type called `iphone:tweet`. By mapping events to synthetic aggregates of coarser granularity, the number of event types is reduced.
2. *Select a set of event types.* Users are required to pick a set of events to be included in the analysis, which can be custom event types from above. This also has the effect of limiting the number of event types.
3. *Pick an alignment point and duration (n events) before or after the alignment point.* By explicitly specifying which part of the session to analyze, we can limit the sequence length to $\leq n$.

5.2 System Description

Ideally, we wish to provide an entirely interactive experience for exploring session sequences. However, changing event mappings requires re-processing all raw session sequences (>100 million rows per day), which is computationally expensive. Our solution was to split the system into two components: The first part is *data pre-processing*, where we designed a set of Hadoop data transformations and a simple GUI for specifying those transformations. The second part is the *visualization tool* to display results from the first step. Using this workflow, users first submit a Hadoop job, wait for the job to complete, and then explore results using the visualization tool.

5.2.1 Data Pre-processing

Step 1: Dynamic event definition. Users first need to define a *dictionary*, i.e., a mapping from raw events to custom events, which also specifies the list of events that will be included in the analysis. A dictionary consists of multiple rules, where each rule specifies a mapping from a regular expression of raw event names to custom event names. Multiple rules can map to the same custom event. A unique unicode character encoding is automatically generated for each custom event. To resolve raw event names that match multiple rules, we take advantage of rule order and apply only the first match. For example, consider a sample dictionary rule below:

```
'web:.*:.*:.*:.*:tweet' => 'tweet'
```

Any event that matches the regex `web:.*:.*:.*:.*:tweet` will be converted to the custom event `tweet`; behind the scenes, this custom event will be stored in the output string as a unicode character (for example, `T` or `\u0054`). Raw events that do not match any rule in the dictionary will be removed during pre-processing, thus reducing the total number of events under consideration and reducing visual complexity downstream. To assist the user in getting started, we have predefined a *default dictionary* that converts raw client events into custom events at the page-level granularity. This default ignores details below the page level and treats all `impression` events as page visits. One common approach is to explicitly track only major pages and group the remaining minor pages into a “wildcard category” (e.g., `web:others`).

```
'web:home:.*:.*:.*:impression'    => 'web:home'
'web:profile:.*:.*:.*:impression' => 'web:profile'
'web:search:.*:.*:.*:impression'  => 'web:search'
...
'web:.*:.*:.*:.*:impression'      => 'web:others'
```

Step 2: Customize alignment, direction, and window size. Users then select an event from the dictionary in the previous step as the alignment point, the “direction” of analysis (before or after the alignment point), and window size (number of events). Hadoop jobs will search for the alignment point in each session and capture a subsequence around the alignment point based on the window size and the direction of analysis. By default, the alignment point is set to the beginning of sessions. It is possible that there are multiple occurrences of the alignment event in a sequence, so users can choose to include all, the first k, or the last k occurrences.

Step 3: Run data transformation job. At this point, a Hadoop data transformation job takes over to process and aggregate the raw data as follows: Each session sequence in the data warehouse is first decoded into a sequence of raw event names. For example, the session sequence `\u0560\u0562\u0564\u0561` would be decoded into `session:start` → `web:home:-:-:-:impression` → `web:home:composer:-:-:tweet` → `session:end`. Each raw event name is mapped to a custom event using the dictionary from Step 1 and converted into a unicode character using new encodings for the custom events. For instance, the example sequence above is converted into a sequence of custom events `start` → `home` → `tweet` → `end` and encoded as `\u0041\u0043\u0054\u0042` or `ACTB`. Next, duplicate consecutive events within each session are removed. For example, the session `AABCCCCABCA` is transformed into `ABCABCA`. After that, the job finds the alignment point(s) and creates window(s) of the appropriate size. For instance, using `B` as the alignment event and a window size of three (after the alignment), the sequence `ABCABAD` is transformed into two windows `BCA` and `BAD`. The data transformation job then takes the union all windows from all sequences and performs a group by to count occurrences.

$$\{(BCA),(BCA),(BAD)\} \rightarrow \{(BCA,2),(BAD,1)\} = sequences$$

If the number of unique sequences ($Q = n(sequences)$) is small enough, the data pre-processing can end here and users can view the results, but this is often not the case. Long and infrequent patterns are the main issues. In the example below, two sequences `ABCDF` and `ABCDG` occur only once. In reality, an analysis might generate thousands of single-occurrence sequences.

```
var sequences = [ {pattern: 'ABC',   count: 2000},
                  {pattern: 'ABCD',  count: 80},
                  {pattern: 'ABCE',  count: 20},
                  {pattern: 'ABCDF', count: 1},
                  {pattern: 'ABCDG', count: 1}    ];
```

These rare sequences comprise a significant portion of typical results and cannot simply be ignored. To retain them but reduce the number of unique sequences, we developed a heuristic `squeezeSequences` algorithm to truncate rare sequences and aggregate them based on a *minimum frequency threshold*:

```
var MIN_FREQ_THRESHOLD = 100, TRUNCATION_MARK = 'x';
function truncate(sequence){
  if(sequence.pattern.lastChar()!=TRUNCATION_MARK){
    // 'ABCDF' => 'ABCDx'
    sequence.pattern = sequence.pattern
      .replaceLastChar(TRUNCATION_MARK);
  } else if(sequence.pattern.length>=2){
    // 'ABCDx' => 'ABCx'
    sequence.pattern = sequence.pattern
      .substring(0, sequence.pattern.length-2)
      + TRUNCATION_MARK
  }
}
function truncateAndRecount(sequences){
  var frequent = new array(), rare = new array();
  for(seq in sequences){
    if(seq.count >= MIN_FREQ_THRESHOLD){
      frequent.add(seq);
    } else{ rare.add(seq); }
  }
  for(seq in rare){ truncate(seq); }
  return groupAndCount(frequent.concat(rare));
}
function squeezeSequences(sequences){
  var round = 0;
  while(sequences.length>=UNIQUE_SEQUENCES_THRESHOLD
    && round<WINDOW_SIZE){
      sequences = truncateAndRecount(sequences);
      round++;
  }
  return sequences;
}
```

Empirically, we have found that this algorithm works well. After processing the data in this manner, the output is ready to be visualized. Via the data transformation process, hundreds of millions of session sequences (terabytes) are summarized into a small data file (typically <15MB), which can be stored in MySQL or simply as a plain CSV file to serve as input to the visualization interface.

5.2.2 Visualization tool

Our visualization tool, called *Flying Sessions*, is shown in Figure 8. It was implemented using D3.js [3] and Ruby on Rails. The system is a simplified version of LifeFlow with two main differences: First, the time dimension is removed, and thus the visual display is a standard icicle tree. Second, the layout is changed from left-to-right to top-to-bottom to make it easier to render labels.

The control panel on the left allows users to filter the sessions based on a few dimensions: user type, platform, experiment bucket, etc. Users can enable/disable events to hide them from the view: these are the custom set of events that users have defined in the data transformation step (not the raw events). Users can also change the colors associated with these events.

On the right, the results are visualized as an icicle tree. Each rectangle represents an event and its color encodes the event type. The width of each rectangle encodes the number of sessions. If the rectangle is wide enough, the label for the event type is displayed. Future events are placed under previous events, so the entire interface flows downward temporally. Users can click on any rectangle to zoom into the subsequence after the clicked rectangle (event).

5.3 Case studies

Flying Sessions was deployed for internal use in January 2013, with detailed documentation. Due to its complexity, it attracted a smaller group of users compared to Scribe Radar, but the users were willing to spend more time learning the tool. For some users, we helped them run their analyses and analyzed the results together. In other cases, we provided technical support in the beginning and let users perform additional analyses by themselves. Summarizing our experiences, a few representative use cases are as follows:

Case 1: What do users do when they first visit Twitter? We believe that users' first actions when they visit Twitter are predictive of overall engagement, and thus we wanted to explore the early portions of session sequences. To accomplish this, we defined a dictionary that mapped raw events to page-level granularity (`web:home, web:profile, web:search, web:connect, etc.`) and examined ten events from the beginning of sessions (`session-start`).

The resulting visualization clearly shows the fraction of traffic to each page. From the interface, we can identify common navigation patterns that correspond to how users engage with different product features. For example, we can see the number and fraction of sessions that begin with a visit to the home timeline (`session-start` → `web:home`). We can identify which pages users navigated from the home timeline to next. The interface allows us to drill down and examine sessions with a specific pattern, e.g., a visit to the home timeline followed by a visit to the profile page (`session-start` → `web:home` → `web:profile`). Overall, Flying Sessions provides a high-level understanding of where and how people begin spending time on Twitter.

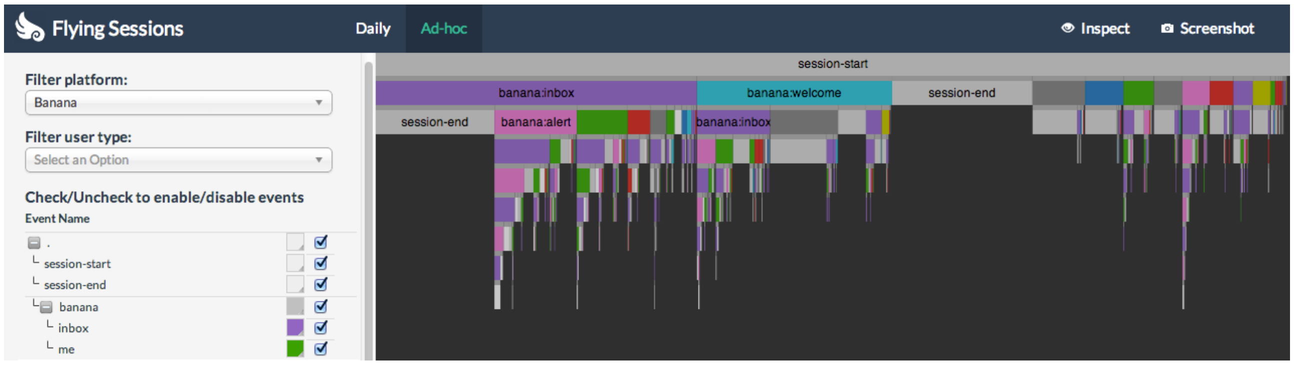
This particular analysis has proven to be sufficiently useful that we have deployed an automatically scheduled task that recomputes the latest data on a daily basis. Figure 8 demonstrates the same use case applied to a hypothetical product.

Case 2: How did the introduction of a new feature change user behavior? Flying Sessions can be used to understand the impact of a particular feature. We illustrate with one example: there was an experiment that changed the design of the logged out home page to include a new feature. Let's call it feature X. The explicit goal was to encourage users to try X and engage with the product. The engineers who ran the experiment used our tool to analyze the results: a visit to the logged out home page (`web:front`) was used as the alignment point, with a window size of ten following events. The custom dictionary included a client event that indicates the use of feature X (`web:x`) and the rest are page-level events (`web:home, web:search, web:others`) from the default dictionary.

Results showed that a small percentage of users invoked feature X after their visit to the logged out home page (`web:front` → `web:x`). However, comparing results between the control and the treatment group, it was discovered that the number of users who then log in from the new logged out home page dropped significantly (`web:front` → `web:home`). The engineers realized that the new design inadvertently made the log in box less prominent on the page, an insight made possible by our visualization.

Case 3: Where do follows happen? The "follow" relationship defines the Twitter interest graph, and the creation of additional edges in this network is one of the most important actions on Twitter since denser connections lead to more user engagement. There are multiple locations in the interface where users can perform the follow action, and we would like to better understand follow behavior.

For this analysis, the custom dictionary was modified from the default dictionary to provide finer-grained details. Because the profile page (`web:profile`) has a few sections related to follows, we broke the `web:profile` event down into section-level events:



Filter platform: Banana

Filter user type: Select an Option

Check/Uncheck to enable/disable events

Event Name		
.		✓
session-start		✓
session-end		✓
banana		✓
inbox	🟪	✓
me	🟩	✓

Fig. 8. Flying Sessions — This screenshot shows the beginnings of user sessions from a hypothetical product on the *banana* platform. We see that 30% of users started in the `inbox` (*purple*); some of these sessions continued with a series of back and forth visits to `alert` (*pink*) and `inbox`; another 20% of sessions began at the `welcome` page (*cyan*), 40% of these successfully logged in and moved to the `inbox`. Users can mouse over any rectangle in the display to see more detail via tooltips or click to zoom in.

```
'.*:.*:.*:.*:.*:follow'        =>  'follow'
'web:home:.*:.*:.*:impression' =>  'web:home'
// See a list of followers on the profile page
'web:profile:followers:.*:.*:impression'
  => 'web:profile:followers'
// See a list of followees on the profile page
'web:profile:following:.*:.*:impression'
  => 'web:profile:following'
// See other parts of the profile page
'web:profile:.*:.*:.*:impression'
  => 'web:profile:others'
...
```

This example illustrates the ability of our tool to specify analyses at an arbitrary level of detail. The data transformation was performed using the `follow` event as an alignment point, retaining the previous ten events. The resulting visualization in Flying Sessions shows user paths that led up to the follow: examining these sequences might be useful in improving the overall flow of the interface. For example, we might find that users click on `follow` from page X frequently; furthermore, the visualization might tell us that page Y is actually responsible for driving the traffic to page X. If users navigate from page Y to page X simply to `follow`, it might be a good idea to let users `follow` on page Y directly, thereby creating a useful navigation shortcut.

6 Conclusions and Future Work

This work explores the application of information visualization techniques to data analytics on large-scale event logs. One important characteristic of these data is the multi-level hierarchical event namespace, which unifies logs from multiple platforms to simplify data discovery. Due to this structure, many analyses are possible using only the event names, a feature we exploit via session sequences.

Taking advantage of the multi-level event hierarchy, we describe an interactive visualization called *Scribe Radar*, built on top of pre-processed event counts from Hadoop. Scribe Radar facilitates the data identification process and improves how data scientists search for events related to their analyses. In addition, the tool provides engineers visibility into data quality issues and assists in detecting data anomalies. We also found Scribe Radar to be useful for understanding the impact of product features after their introduction.

We bring visualization and large-scale funnel analysis together in *Flying Sessions*. Users can specify the granularity of their analyses by defining custom dictionaries and selecting parts of the sessions. Hadoop then handles data transformations to provide users aggregated results that can be explored interactively in a visualization interface.

The biggest challenge when designing these systems is defining appropriate intermediate data, as illustrated in Figure 9. Because the magnitude of raw data is beyond the capabilities of standard client machines and many traditional databases, we must reduce intermediate data size prior to visualization. This is usually accomplished via data aggregation, which may limit the user's ability to explore the data since the dimension of interest might have been eliminated via the aggregation. Providing richer methods for data exploration on the client side requires more detail and thus more intermediate data. One approach that has worked for us is to use traditional DBMS technologies to hold intermediate data (beyond what can be held in memory inside the browser); the client application can then query for relevant data and perform additional online processing as needed.

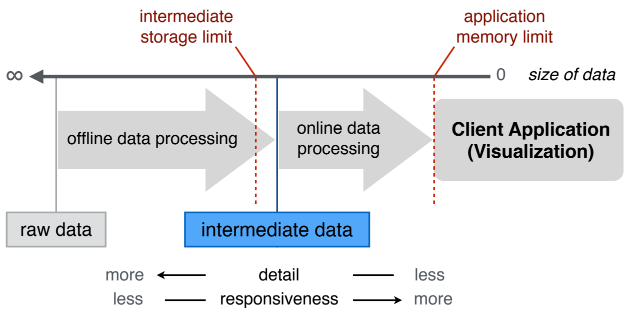


Fig. 9. Illustration of the tradeoff between responsiveness and detail as encoded in the intermediate data.

Although our techniques directly exploit a hierarchical event namespace, we believe that they can be generalized and applied to different applications. While scale is certainly interesting, our techniques are equally applicable to cases where the data are small enough to be processed on a single machine or even directly within the browser.

While the case studies and user feedback have shown that our techniques are promising, there is still plenty of room for improvement. Event monitoring would become more convenient and effective with the inclusion of anomaly detection and a system for automatic alerts—in general, we see great synergy in the integration of both visualization and data mining techniques. Another area of work is to provide more interactivity and reduce iteration time in funnel analysis, since the current setup still requires batch processing on the cluster. We envision that Flying Sessions could operate in *preview* mode on samples of session sequences in a truly interactive fashion, and once users are satisfied with the previewed results, they can submit a full job to process all data. Sampling may provide an effective strategy to balance the demands of interactivity and the need for large-scale batch processing. We hope that our work will inspire further research at the intersection of visualization and large-scale data analysis.

Acknowledgments

None of the work described in this paper would have been possible without the engineers and data scientists at Twitter. In particular, we would like to thank Linus Lee, Chuang Liu, and the visual-insights team. We also appreciate comments from VAST reviewers and feedback from Ben Shneiderman and Catherine Plaisant.

References

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In *International Conference on Data Engineering (ICDE)*, pages 3–14, 1995.

[2] R. Bade, S. Schlechtweg, and S. Miksch. Connecting time-oriented data and information to a coherent interactive visualization. In *ACM CHI Conference on Human Factors in Computing Systems*, pages 105–112, 2004.

[3] M. Bostock, V. Ogievetsky, and J. Heer. D3: Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2301–2309, 2011.

[4] M. Burch, F. Beck, and S. Diehl. Timeline trees: Visualizing sequences of transactions in information hierarchies. In *Working Conference on Advanced Visual Interfaces (AVI)*, pages 75–82, 2008.

[5] I. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White. Visualization of navigation patterns on a web site using model-based clustering. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 280–284, 2000.

[6] E. H. Chi. Improving web usability through visualization. *IEEE Internet Computing*, 6(2):64–71, 2002.

[7] J. Dean and S. Ghemawat. MapReduce: Simplified data processing on large clusters. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2004.

[8] J. Fails, A. Karlson, L. Shahamat, and B. Shneiderman. A visual interface for multivariate temporal data: Finding patterns of events across multiple histories. In *IEEE Symposium on Visual Analytics Science and Technology (VAST)*, pages 167–174, 2006.

[9] J.-D. Fekete. The InfoVis Toolkit. In *IEEE Symposium on Information Visualization (InfoVis)*, pages 167–174, 2004.

[10] M. Ghoniem, G. Shurkhovetskyy, A. Bahey, and B. Otjacques. VAFLE: Visual analytics of firewall log events. In *Visualization and Data Analysis*, 2013.

[11] M. Gray, A. Badre, and M. Guzdial. Visualizing usability log data. In *IEEE Symposium on Information Visualization (InfoVis)*, pages 93–98, 1996.

[12] J. Guerra-Gómez, M. L. Pack, C. Plaisant, and B. Shneiderman. Visualizing change over time using dynamic hierarchies: TreeVersity2 and the StemView. *IEEE Transactions on Visualization and Computer Graphics*, 19(12):2566–2275, Dec. 2013.

[13] B. L. Harrison, R. Owen, and R. M. Baecker. Timelines: An interactive system for the collection and visualization of temporal data. In *Graphics Interface (GI)*, pages 141–148, 1994.

[14] H. Hochheiser and B. Shneiderman. Using interactive visualizations of WWW log data to characterize access patterns and inform site design. *Journal of the American Society for Information Science and Technology*, 52(4):331–343, 2001.

[15] J. I. Hong and J. A. Landay. WebQuilt: A framework for capturing and visualizing the web experience. In *International Conference on World Wide Web (WWW)*, pages 717–724, 2001.

[16] B. Johnson and B. Shneiderman. Tree-Maps: A space-filling approach to the visualization of hierarchical information structures. In *IEEE Conference on Visualization (VIS)*, pages 284–291, 1991.

[17] S. Kandel, J. Heer, C. Plaisant, J. Kennedy, F. van Ham, N. H. Riche, C. Weaver, B. Lee, D. Brodbeck, and P. Buono. Research directions in data wrangling: Visualizations and transformations for usable and credible data. *Information Visualization*, 10(4):271–288, Sept. 2011.

[18] G. M. Karam. Visualization using timelines. In *ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 125–137, 1994.

[19] D. A. Keim, F. Mansmann, J. Schneidewind, and H. Ziegler. Challenges in visual data analysis. In *International Conference on Information Visualization (IV)*, pages 9–16, 2006.

[20] J. B. Kruskal and J. M. Landwehr. Icicle plots: Better displays for hierarchical clustering. *The American Statistician*, 37(2):162–168, 1983.

[21] H. Lam, D. Russell, D. Tang, and T. Munzner. Session viewer: Visual exploratory analysis of web session logs. In *IEEE Symposium on Visual Analytics Science and Technology (VAST)*, pages 147–154, 2007.

[22] J. Lamping and R. Rao. The Hyperbolic Browser: A focus+context technique for visualizing large hierarchies. *Journal of Visual Languages & Computing*, 7(1):33–55, Mar. 1996.

[23] G. Lee, J. Lin, C. Liu, A. Lorek, and D. Ryaboy. The unified logging infrastructure for data analytics at Twitter. In *International Conference on Very Large Data Bases (VLDB)*, pages 1771–1780, 2012.

[24] T. Mah, H. Hoek, and Y. Li. Funnel report mining for the MSN network. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 450–455, 2001.

[25] M. Monroe, R. Lan, H. Lee, C. Plaisant, and B. Shneiderman. Temporal event sequence simplification. *IEEE transactions on visualization and computer graphics*, 19(12):2227–2236, Dec. 2013.

[26] T. Munzner. H3: Laying out large directed graphs in 3D hyperbolic space. In *Visualization Conference, Information Visualization Symposium and Parallel Rendering Symposium (VIZ)*, pages 2–10, 1997.

[27] T. Munzner, F. Guimbretière, S. Tasiran, L. Zhang, and Y. Zhou. TreeJuxtaposer: Scalable tree comparison using focus+context with guaranteed visibility. *ACM Transactions on Graphics*, 22(3):453–462, 2003.

[28] D. Phan, A. Paepcke, and T. Winograd. Progressive multiples for communication-minded visualization. In *Graphics Interface (GI)*, pages 225–232, 2007.

[29] J. E. Pitkow and K. A. Bharat. Webviz: A tool for world wide web access log analysis. In *International Conference on World Wide Web (WWW)*, 1994.

[30] C. Plaisant, R. Mushlin, A. Snyder, J. Li, D. Heller, and B. Shneiderman. LifeLines: Using visualization to enhance navigation and analysis of patient records. In *AMIA Annual Symposium*, pages 76–80, 1998.

[31] E. M. Reingold and J. S. Tilford. Tidier Drawings of Trees. *IEEE Transactions on Software Engineering*, SE-7(2):223–228, Mar. 1981.

[32] G. G. Robertson, S. K. Card, and J. D. Mackinlay. Information visualization using 3D interactive animation. *Communications of the ACM*, 36(4):57–71, 1993.

[33] Z. Shen, J. Wei, N. Sundaresan, and K.-L. Ma. Visual analysis of massive web session data. In *IEEE Symposium on Large Data Analysis and Visualization (LDAV)*, pages 65–72, Oct. 2012.

[34] J. Stasko and E. Zhang. Focus+context display and navigation techniques for enhancing radial, space-filling hierarchy visualizations. In *IEEE Symposium on Information Visualization (InfoVis)*, pages 57–65, 2000.

[35] T. Takada and H. Koike. Mielog: A highly interactive visual log browser using information visualization and statistical analysis. In *USENIX Conference on System Administration (LISA)*, pages 133–144, 2002.

[36] A. Thusoo, Z. Shao, S. Anthony, D. Borthakur, N. Jain, J. Sen Sarma, R. Murthy, and H. Liu. Data warehousing and analytics infrastructure at Facebook. *International Conference on Management of Data (SIGMOD)*, pages 1013–1020, 2010.

[37] Y. Tu and H.-W. Shen. Visualizing changes of hierarchical data using treemaps. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1286–1293, 2007.

[38] K. Vrotsou, J. Johansson, and M. Cooper. ActiviTree: Interactive visual exploration of sequences in event-based data using graph similarity. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):945–52, 2009.

[39] T. D. Wang, C. Plaisant, A. J. Quinn, R. Stanchak, S. Murphy, and B. Shneiderman. Aligning temporal data by sentinel events: Discovering patterns in electronic health records. In *ACM CHI Conference on Human Factors in Computing Systems*, pages 457–466, 2008.

[40] M. Wattenberg. Visualizing the stock market. In *ACM CHI Conference on Human Factors in Computing Systems*, pages 188–189, 1999.

[41] J. Wei, Z. Shen, N. Sundaresan, and K.-L. Ma. Visual cluster exploration of web clickstream data. In *IEEE Conference on Visual Analytics Science and Technology (VAST)*, pages 3–12, 2012.

[42] C. Wetherell and A. Shannon. Tidy drawings of trees. *IEEE Transactions on Software Engineering*, SE-5(5):514–520, Sept. 1979.

[43] K. Wongsuphasawat and D. Gotz. Exploring flow, factors, and outcomes of temporal event sequences with the outflow visualization. *IEEE Transactions on Visualization and Computer Graphics*, 18(12):2659–2668, Dec. 2012.

[44] K. Wongsuphasawat, J. A. Guerra Gómez, C. Plaisant, T. D. Wang, M. Taieb-Maimon, and B. Shneiderman. LifeFlow: Visualizing an overview of event sequences. In *ACM CHI Conference on Human Factors in Computing Systems*, pages 1747–1756, 2011.

[45] K. Wongsuphasawat, C. Plaisant, M. Taieb-Maimon, and B. Shneiderman. Querying event sequences by exact match or similarity search: Design and empirical evaluation. *Interacting with computers*, 24(2):55–68, Mar. 2012.