Analyzing Log Analysis: An Empirical Study of User Log Mining

S. Alspaugh, *University of California, Berkeley and Splunk Inc.;* Beidi Chen and Jessica Lin, *University of California, Berkeley;* Archana Ganapathi, *Splunk Inc.;* Marti A. Hearst and Randy Katz, *University of California, Berkeley*

https://www.usenix.org/conference/lisa14/conference-program/presentation/alspaugh

This paper is included in the Proceedings of the 28th Large Installation System Administration Conference (LISA14).

November 9–14, 2014 • Seattle, WA

ISBN 978-1-931971-17-1

Open access to the Proceedings of the 28th Large Installation System Administration Conference (LISA14) is sponsored by USENIX

Analyzing Log Analysis: An Empirical Study of User Log Mining

S. Alspaugh* *UC Berkeley*
Beidi Chen *UC Berkeley*
Jessica Lin *UC Berkeley*
Archana Ganapathi *Splunk Inc.*
Marti A. Hearst *UC Berkeley*
Randy Katz *UC Berkeley*

Abstract

We present an in-depth study of over 200K log analysis queries from Splunk, a platform for data analytics. Using these queries, we quantitatively describe log analysis behavior to inform the design of analysis tools. This study includes state machine based descriptions of typical log analysis pipelines, cluster analysis of the most common transformation types, and survey data about Splunk user roles, use cases, and skill sets. We find that log analysis primarily involves filtering, reformatting, and summarizing data and that non-technical users increasingly need data from logs to drive their decision making. We conclude with a number of suggestions for future research.

Tags: log analysis, query logs, user modeling, Splunk, user surveys

1 Introduction

Log analysis is the process of transforming raw log data into information for solving problems. The market for log analysis software is huge and growing as more business insights are obtained from logs. Stakeholders in this industry need detailed, quantitative data about the log analysis process to identify inefficiencies, streamline workflows, automate tasks, design high-level analysis languages, and spot outstanding challenges. For these purposes, it is important to understand log analysis in terms of discrete tasks and data transformations that can be measured, quantified, correlated, and automated, rather than qualitative descriptions and experience alone.

This paper helps meet this need using over 200K queries recorded from a commercial data analytics system called Splunk. One challenge is that logged system events are not an ideal representation of human log analysis activity [3]. Logging code is typically not designed to capture human behavior at the most efficacious level of granularity. Even if it were, recorded events may not reflect internal mental activities. To help address this gap, we supplement the reported data with results of a survey of Splunk sales engineers regarding how Splunk is used in practice.

In our analysis, we examine questions such as: What transformations do users apply to log data in order to analyze it? What are common analysis workflows, as described by sequences of such transformations? What do such workflows tell us, qualitatively, about the nature of log analysis? Who performs log analysis and to what end? What improvements do we need to make to analysis tools, as well as to the infrastructure that logs activities from such tools, in order to improve our understanding of the analysis process and make it easier for users to extract insights from their data?

The answers to these questions support a picture of log analysis primarily as a task of filtering, reformatting, and summarizing. Much of this activity appears to be data munging, supporting other reports in the literature [28]. In addition, we learn from our survey results that users outside of IT departments, including marketers and executives, are starting to turn to log analysis to gain business insights. Together, our experience analyzing these queries and the results of our analysis suggest several important avenues for future research: improving data transformation representation in analytics tools, implementing integrated provenance collection for user activity record, improving data analytics interfaces and creating intelligent predictive assistants, and further analyzing other data analysis activities from other systems and other types of data besides logs.

*This author was an employee of Splunk Inc. when this paper was written.

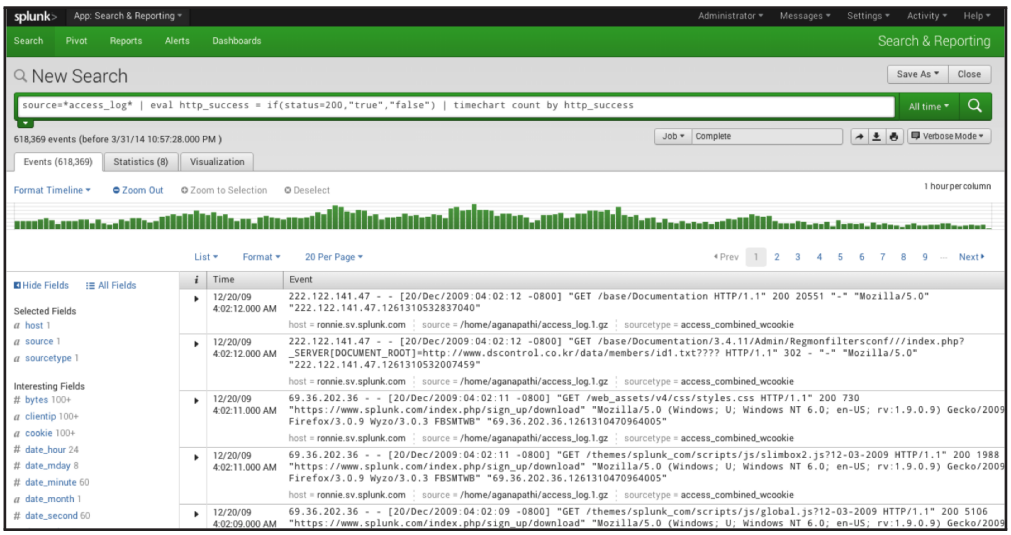


Figure 1: The default Splunk GUI view displays the first several events indexed, with extracted fields highlighted on the side, and a histogram of the number of events over time displayed along the top. The user types their query into the search bar at the top of this view.

2 Related Work

We discuss (1) systems for log analysis, (2) techniques for log analysis, and (3) results of log analysis, so that those log analysis activities can be compared to our observations. We also discuss (4) user studies of system administrators – one of the primary classes of log analysts – and (5) of search engine users – where query logs are the main source of data on user behavior and needs.

Systems for log analysis: The purpose of this section is not to compare Splunk to other analysis systems, but to describe the uses these systems support, to provide a sense of how our observations fit within the larger context. Dapper, Google's system tracing infrastructure, is used by engineers to track request latency, guarantee data correctness, assess data access costs, and find bugs [34]. From their detailed descriptions, we can infer that engineers use transformations similar to those used by Splunk users. Other systems, such as Sawzall and PigLatin, include query languages that extract data from logs with heavy use of these same types of transformations [30, 26]. These points suggest that the activity records we have collected may represent typical log analysis usage, despite being gathered from only one system.

Techniques for log analysis: Published techniques for log analysis center around the main challenges in working with logs, such as dealing with messy formats, and solving event-based problems [23]. This includes event and host clustering [20, 21], root failure diagnosis [8, 17], anomaly detection [18], dependency inference [25, 19], and data extraction [16, 39]. Although their motivating use cases overlap with Splunk use cases, in our observations, the use of such techniques appears to be relatively rare (even though Splunk does provide, e.g., clustering and anomaly detection functionality).

Results of log analysis: Log analysis is also used in research as a means to an end rather than as the subject itself. Logs have been used to explain system behavior [7, 6], understand failures [31, 24], identify design flaws [11], spot security vulnerabilities [15], highlight new phenomena [29], and drive system simulations [12]. To the extent that such research involves heavy application of human inference rather than "automatic" statistical inference techniques, like many of those mentioned in the previous section, it appears to more closely align with our observations of log analysis behavior. However, the problems addressed are naturally of an academic nature, whereas Splunk users of often looking for timely business insights specific to their situation.

System administrator user studies: As system administrators are one of the primary classes of log analysts, studies of their behavior are relevant to our study of log analysis. Researchers have studied system administrators to characterize their work environments and problems commonly faced [4], as well as the mental models they form [13]. One study surveying 125 system administrators discovered that accuracy, reliability, and credibility are considered the most important features in tools [38]. Other researchers have called for more standardization in system administration activities – such efforts will benefit from the data we present [9].

Term	*Definition*
event	a raw, timestamped item of data indexed by Splunk, similar to a tuple or row in databases
field	a key corresponding to a value in an event, similar to the concept of a column name
value	part of an event corresponding to a certain field, similar to a particular column entry in a particular row
query	a small program written in the Splunk query language, consisting of pipelined stages
stage	a portion of a query syntactically between pipes; conceptually a single transformation
transformation	an abstract category of similar commands e.g., filter or aggregate; each stage is a transformation
command	the part of a stage that indicates what operation to apply to the data
argument	the parts of a stage that indicate what fields, values, or option values to use with a command
interactive	a query that is run when it is entered by the user into the search bar
scheduled	a query that has been saved by a user and scheduled to run periodically like a `cron` job

Table 1: Terminology describing Splunk data.

Search engine query log studies: While we are unaware of prior work that uses query logs to study *analysis* behavior, query logs are often used to study *search engine user* behavior. People have used search engine query logs to model semantic relationships [22], track user preferences [35], and identify information needs [32]. Techniques involve examining query terms and analyzing user sessions [14, 33]. Due to data quality issues discussed in Section 4, we could not analyze user sessions, but other aspects of our current and previous work parallel these techniques [2]. Employing some of these techniques to examine data analysis activity logs is a promising avenue of future research. Going forward we expect that the study of human information seeking behavior will be enriched through the study of analysis query logs.

3 Splunk Logs and Queries

We collected queries from Splunk[1], a platform for indexing and analyzing large quantities of data from heterogeneous data sources, especially machine-generated logs. Splunk is used for a variety of data analysis needs, including root cause failure detection, web analytics, A/B testing and product usage statistics. Consequently, the types of data sets indexed in Splunk also span a wide range, such as system event logs, web access logs, customer records, call detail records, and product usage logs. This section describes the Splunk data collection and query language in more detail; Table 1 lists the terminology introduced in this section.

3.1 Overview

Data collection To use Splunk, the user indicates the data that Splunk must index, such as a log directory on a file system. Splunk organizes this data into temporal *events* by using timestamps as delineators, and processes these events using a MapReduce-like architecture [5]. Splunk does not require the user to specify a schema for the data, because much log data is semi-structured or unstructured, and there is often no notion of a schema that can be imposed on the data a priori. Rather, *fields* and *values* are extracted from events at run time based on the *source type*. Specifically, when a user defines a new source type, Splunk guides the user in constructing regular expressions to extract fields and values from each incoming raw event.

Query llanguage Splunk includes a query language for searching and manipulating data and a graphical user interface (GUI) with tools for visualizing query results. The query consists of a set of *stages* separated by the pipe character, and each stage in turn consists of a *command* and *arguments*. Splunk passes events through each stage of a query. Each stage filters, transforms or enriches data it receives from the previous stage, and pipes it to the subsequent stage, updating the displayed results as they are processed. A simple example of a query is a plain text search for specific strings or matching field-value pairs. A more complex example can perform more advanced transformations, such as clustering the data using k-means. Users can save certain queries and schedule them to be run on a given schedule, much like a `cron` job. We call these queries *scheduled* queries.

Graphical user interface Users almost always compose Splunk queries in the GUI. The default GUI view displays the first several events indexed, with extracted fields highlighted on the left hand side, and a histogram of the number of events over time displayed along the top. A screen shot of this default view is shown in Figure 1. The user types their query into the search bar at the top of this view. When the user composes their query in the GUI, we call it an *interactive* query.

When the user enters a query that performs a filter, the GUI updates to display events which pass through the filter. When the user uses a query to add or transform a field, the GUI displays events in updated form. Most queries result in visualizations such as tables, time series, and histograms, some of which appear in the GUI when the query is executed, in the "Visualization" tab (Fig-

[1] `www.splunk.com`

ure 1). Users can also create "apps," which are custom views that display the results of pre-specified queries, possibly in real time, which is useful for things like monitoring and reporting. Although the set of visualizations Splunk offers does not represent the full breadth of all possible visualizations, they still capture a large set of standard, commonly used ones.

3.2 An Example Splunk Query

The Splunk query language is modeled after the Unix `grep` command and pipe operator. Below is an example query that provides a count of errors by detailed status code:

```
search error | stats count by status | lookup
    statuscodes status OUTPUT statusdesc
```

This example has three stages: `search`, `stats`, and `lookup` are the commands in each stage, `count by` and `OUTPUT` are functions and option flags passed to these commands, and "error", "status", "statuscodes", and "statusdesc" are arguments. In particular, "status" and "statusdesc" are fields.

To see how this query operates, consider the following toy data set:

0.0	-	**error**	404
0.5	-	OK	200
0.7	-	**error**	500
1.5	-	OK	200

The first stage of the query (`search` error) filters out all events not containing the word "error". After this stage, the data looks like:

0.0	-	**error**	404
0.7	-	**error**	500

The second stage (`stats count by` status) aggregates events by applying the `count` function over events grouped according to the "status" field, to produce the number of events in each "status" group.

count	status
1	404
1	500

The final stage (`lookup` status codes status `OUTPUT` statusdesc) performs a join on the "status" field between the data and an outside table that contains descriptions of each of the codes in the "status" field, and puts the corresponding descriptions into the "statusdesc" field.

count	status	statusdesc
1	404	Not Found
1	500	Internal Server Error

Total queries	203691
Interactive queries	18872
Scheduled queries	184819
Distinct scheduled queries	17085

Table 2: Characteristics of the set of queries analyzed from the Splunk logs.

4 Study Data

We collected over 200K Splunk queries. The data set consists of a list of timestamped query strings. Table 2 summarizes some basic information about this query set.

We wrote a parser for this query language; the parser is freely available [2]. This parser is capable of parsing over 90% of all queries in the data set, some of which may be valid failures, as the queries may be malformed. (This limitation only affects the cluster analysis in Section 6.)

It is important to note that we do not have access to any information about the data over which the queries were issued because these data sets are proprietary and thus unavailable. Having access only to query logs is a common occurrence for data analysis, and methodologies that can work under these circumstances are therefore important to develop. Further, by manually inspecting the queries and using them to partially reconstruct some data sets using the fields and values mentioned in the queries, we are fairly certain that these queries were issued over many different sources of data (e.g., web server logs, security logs, retail transaction logs, etc.), suggesting the results presented here will generalize across different datasets.

It is also important to note that some of the queries labeled as interactive in our data set turned out to be programmatically issued from sources external to Splunk, such as a user-written script. It is difficult to separate these mislabeled queries from the true interactive queries, so we leave their analysis to future work, and instead focus our analysis in this paper on scheduled queries.

[2] `https://github.com/salspaugh/splparser`

5 Transformation Analysis

The Splunk query language is complex and supports a wide range of functionality, including but not limited to: reformatting, grouping and aggregating, filtering, reordering, converting numerical values, and applying data mining techniques like clustering, anomaly detection, and prediction. It has 134 distinct core commands at the time of this writing, and commands are often added with each new release. In addition, users and Splunk app developers can define their own commands.

We originally attempted to analyze the logs in terms of command frequencies, but it was difficult to generalize from these in a way that is meaningful outside of Splunk [1]. So, to allow for comparisons to other log analysis workflows and abstract our observations beyond the Splunk search language, we manually classified these 134 commands into 17 categories representing the types of transformations encoded, such as filtering, aggregating, and reordering (Table 3).

Note that because some Splunk commands are overloaded with functionality, several commands actually perform multiple types of transformations, such as aggregation followed by renaming. In these cases, we categorized the command according to its dominant use case.

We use this categorization scheme to answer the following questions about log analysis activity:

- How are the individual data transformations statistically distributed? What are the most common transformations users perform? What are the least common?
- How are sequences of transformations statistically distributed? What type of transformations do queries usually start with? What do they end with? What transformations typically follow a given other transformation?
- How many transformations do users typically apply in a given query? What are the longest common subsequences of transformations?

5.1 Transformation Frequencies

We first counted the number of times that each transformation was used (Figure 2). The most common are **Cache** (27% of stages), **Filter** (26% of stages), **Aggregate** (10% of stages), **Macro** (10% of stages),and **Augment** (9% of stages). Scheduled queries are crafted and set up to run periodically, so the heavy use of caching and macros is unsurprising: Splunk adds caching to scheduled queries to speed their execution, and macros capture common workflows, which are likely to be discovered by users after the iterative, ad hoc querying that results in a "production-ready" scheduled query. Although we do not report directly on them here due to data quality issues (Section 4), anecdotally, it appears that interactive queries have a similar distribution except that the use of **Cache** and **Macro** is less frequent, and the use of **Input** is more frequent.

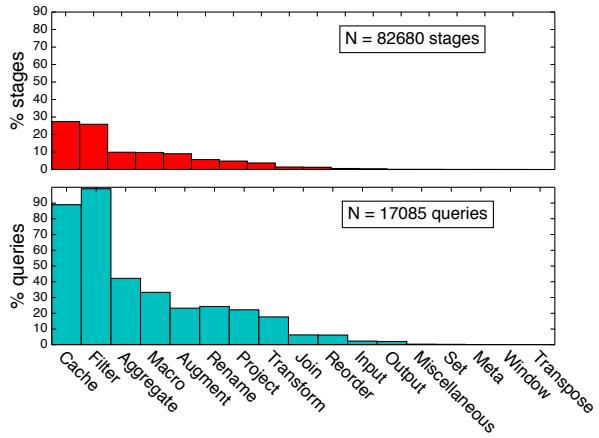


Figure 2: The distribution of data transformations that are used in log analysis. The top graph shows, for each transformation, the percent of stages that apply that transformation. The bottom graph shows, for each transformation, the percent of queries that contain that transformation at least once (so the percents do not add to 100).

For each transformation type, we also computed the number of queries that used that transformation (Figure 2). This gives us some idea of how many of the queries would be expressible in a restricted subset of the language, which is interesting because it tells us the relative importance of various transformations.

From this we see that **Filter** transformations are extremely important – 99% of scheduled queries use such transformations. Without **Aggregate** transformations, 42% of scheduled queries would not be possible. Around a quarter of queries use **Augment**, **Rename**, and **Project** transformations, and 17% use commands that **Transform** columns.

In contrast, **Join**s are only used in 6% of scheduled queries. This possible difference from database workloads could be because log data is not usually relational and generally has no schema, so it may often not have information that would satisfy key constraints needed for join, or it may already be sufficiently denormalized for most uses. It could also be because these are scheduled queries, and expensive **Join** operations have been optimized away, although again anecdotally the interactive queries do not suggest this. **Reorder** transformations are also used only 6% of the time – log events are already ordered by time by Splunk, and this is probably often the desired order. **Input** and **Output** transformations are used in only 2% of scheduled queries – these

Transformation	*Description*	*Top Commands*	*% Queries*	*Examples*

Aggregate	coalesce values of a given field or fields (columns) into one summary value	`stats`
		`timechart`
		`top`
			86.0
			9.0
			3.0
				`stats sum(size_kb)`
				`timechart count by region`
				`top hostname`

Augment	add a field (column) to each event, usually a function of other fields	`eval`
		`appendcols`
		`rex`
			57.0
			19.0
			15.0
				`eval pct=count/total*100`
				`spath input=json`
				`rex "To:  (?<to>.*)"`

Cache	write to or read from cache for fast processing	`summaryindex` `sitimechart`	98.0 30.0	`summaryindex namespace=foo` `sitimechart count by city`

Filter	remove events (rows) not meeting the given criteria	`search`
		`where`
		`dedup`
			100.0
			7.0
			4.0
				`search name="alspaugh"`
				`where count > 10`
				`dedup session_id`

Input	input events into the system from elsewhere	`inputlookup`	88.0	`inputlookup data.csv`
Join	join two sets of events based on matching criteria	`join` `lookup`	82.0 16.0	`join type=outer ID` `lookup`
Macro	apply user-defined sequence of Splunk commands	`‘sourcetype_metrics‘` `‘forwarder_metrics‘`	50.0 13.0	`‘sourcetype_metrics‘` `‘forwarder_metrics‘`
Meta	configure execution environment	`localop`	83.0	`localop`
Miscellaneous	commands that do not fit into other categories	`noop`	39.0	`noop`
Output	write results to external storage or send over network	`outputlookup`		`outputlookup results.csv`
Project	remove all columns except those selected	`table` `fields`	80.0 22.0	`table region total` `fields count`
Rename	rename fields	`rename`	100.0	`rename cnt AS Count`
Reorder	reorder events based on some criteria	`sort`	100.0	`sort - count`
Set	perform set operations on data	`append` `set`	66.0 40.0	`append [...]` `set intersect [...]  [...]`
Transform	mutate the value of a given field for each event	`fillnull` `convert`	96.0 2.0	`fillnull status` `convert num(run_time)`
Transpose	swap events (rows) with fields (columns)	`transpose`	100.0	`transpose`
Window	add fields that are windowing functions of other data	`streamstats`	90.0	`streamstats first(edge)`

Table 3: Manual classification of commands in the Splunk Processing Language into abstract transformations categories. For each transformation category, the *Top Commands* column shows the most-used commands in that category. The *% Queries* column shows, for all queries containing a given transformation, what percent of queries contained that command.

again could have been optimized away, or possibly captured in **Macro**s. Lastly, the other transformations are used in nearly zero queries. In the case of **Window**ing transformations, this could be because windowed operations are accomplished "manually" through sequences of **Augment** transformations or via overloaded commands that were classified as other transformation types. We were surprised such operations were not more common. In the case of the others, such as **Transpose**, it is more likely because log data is rarely of the type for which such operations are needed.

5.2 Transformation Pipelines

Next, for each pair of transformation types, we counted the number of times within a query that the first transformation of the pair was followed by the second transformation of the pair. We used these counts to compute, for each transformation, how frequently each of the other transformation types followed it in a query.

We used these frequencies to create a state machine graph, as shown in Figure 3. Each node is a type of transformation, and each edge from transformation *A* to a transformation *B* indicates the number of times *B* was used after *A* as a fraction of the number of times *A* was used. Also included as nodes are states representing the start of a query, before any command has been issued, and the end of a query, when no further commands are issued. The edges between these nodes can be thought of as transition probabilities that describe how likely a user is to issue transformation *B* after having issued transformation *A*.

Using these graphs, we can discover typical log analysis pipelines employed by Splunk users. We exclude from presentation sequences with **Cache** transformations, as those have in most cases been automatically added to scheduled queries by Splunk to optimize them, as well as **Macro**s, because these can represent any transformation, so we do not learn much by including them. The remaining top transformation pipelines by weight (where the weight of a path is the product of its edges) are:

- **Filter**
- **Filter** | **Aggregate**
- **Filter** | **Filter** [3]
- **Filter** | **Augment** | **Aggregate**
- **Filter** | **Reorder**
- **Filter** | **Augment**

The preponderance of **Filter** transformations in typical pipelines is not surprising given that it is the most frequently applied transformation. It also makes sense in the context of log analysis – logging collects a great deal of information over the course of operation of a system, only a fraction of which is likely to be relevant to a given situation. Thus it is almost always necessary to get rid of this extraneous information. We investigate **Filter**, **Aggregate**, and **Augment** transformations in more detail in Section 6 to explain why these also appear in common pipelines.

These transformations sequences may seem simple compared to some log analysis techniques published in conferences like KDD or DSN [20, 25]. These pipelines more closely correspond to the simpler use cases described in the Dapper or Sawzall papers [34, 30]. There are many possible explanations for this: Most of the problems faced by log analysts may not be data mining or machine learning problems, and when they are, they may be difficult to map to published data mining and machine learning algorithms. Human intuition and domain expertise may be extremely competitive with state of the art machine learning and other techniques for a wide variety of problems – simple filters, aggregations and transformations coupled with visualizations are powerful tools in the hands of experts. Other reasons are suggested by user studies and first-hand industry experience [23, 38]. Users may prefer interpretable, easily adaptable approaches over black-boxes that require lots of mathematical expertise. It is worth further investigating the types of analysis techniques currently in widespread use and assess how the research on analysis techniques can better address practitioner needs.

We hypothesize that one important variable determining what transformation sequences are most often needed is the data type. Thus, we created more focused state machine graphs for two commonly analyzed source types by pulling out all queries that explicitly specified that source type[4]: Figure 4 shows the analysis applied to server access logs, used for web analytics (measuring traffic, referrals, and clicks). Figure 5 shows the results on operating system event logs (analyzing processes, memory and CPU usage). These figures suggest that indeed, query patterns can be expected to differ significantly depending on the type of data being analyzed. This could be due to the domain of the data, which could cause the types of questions asked to vary, or it could be due to the format of the data. For example web logs may have a more regular format, allowing users to avoid the convoluted processing required to normalize less structured data sources.

Other important factors likely include who the user is and what problems they are trying to solve. For example, in

[3] These can be thought of as one **Filter** that happened to be applied in separate consecutive stages.

[4] Source type can be specified in **Filter** transformations – this is what we looked for.

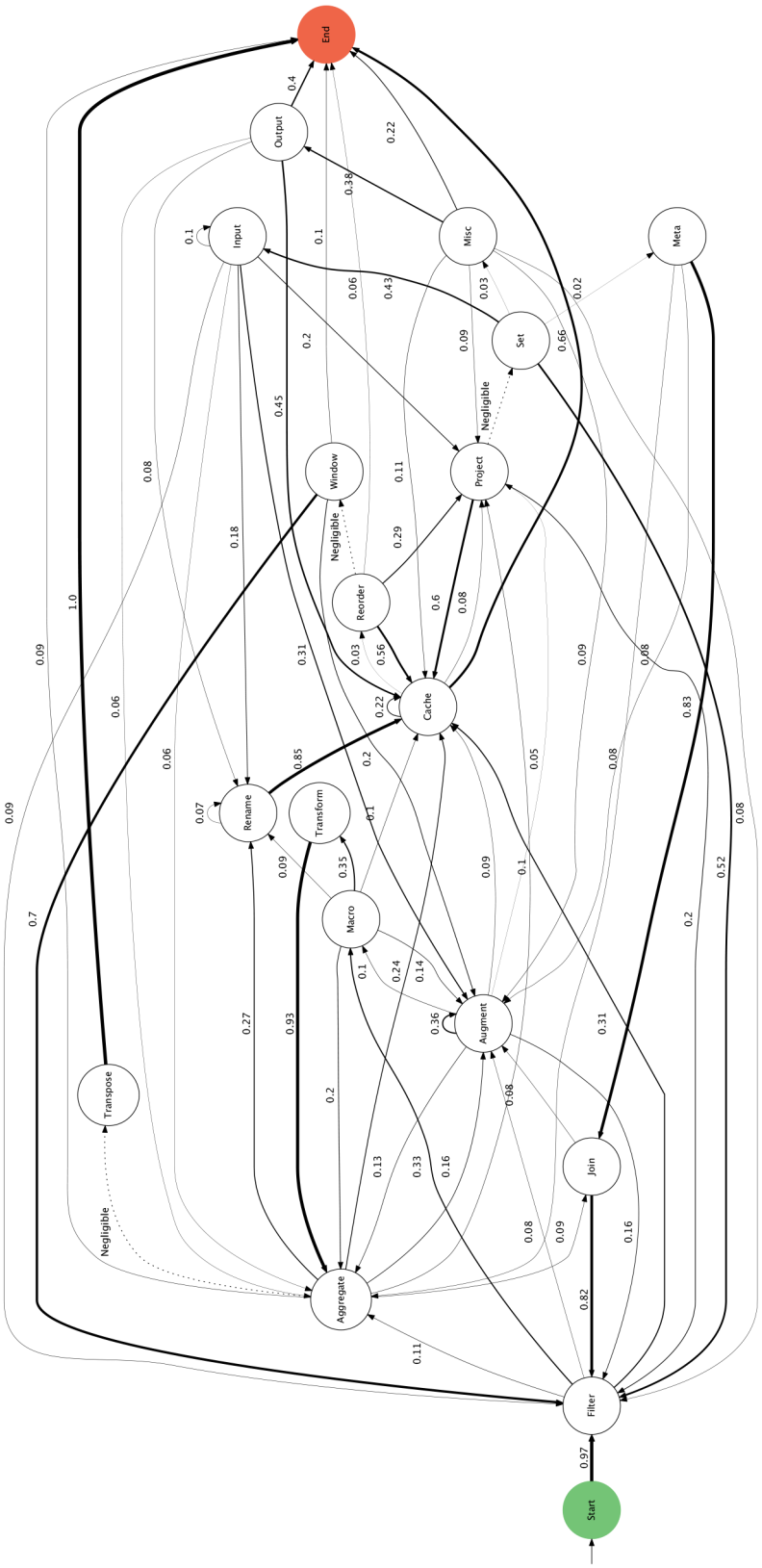
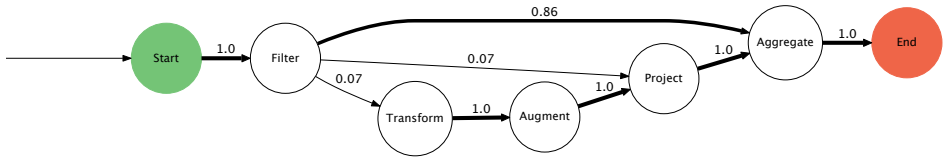
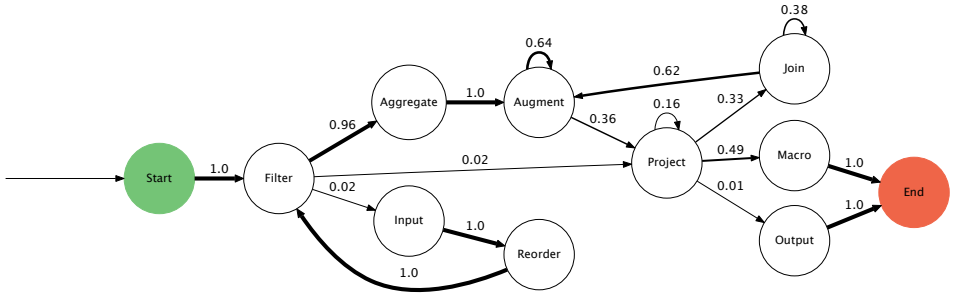


Figure 3: State machine diagram describing, for all distinct scheduled queries, the pairwise transition frequency between the command categories described in the text. Only edges with weight greater or equal to .05 are shown, for clarity.



256 distinct queries

Figure 4: The pairwise transition frequency between transformations for web access log queries.



46 distinct queries

Figure 5: The pairwise transition frequency between transformations for OS event log queries.

the case of web access log data, an operations user will want to know, "Where are the 404s?[5] Are there any hosts that are down? Is there a spike in traffic that I should add capacity for?" A marketer will want to know, "What keywords are people searching today after my recent press release? What are the popular webinars viewed on my website?" A salesperson may ask, "Of the visitors today, how many are new versus returning, and how can I figure out whom to engage in a sales deal next based on what they're looking for on the web site?" Capturing this supplemental information from data analysis tools to include in the analysis would be useful for later tailoring tools to particular use cases. We have gathered some information about this (Section 7) but unfortunately we could not cross-reference this data with query data.

5.3 Longest Subsequences

To investigate what longer, possibly more complex, queries look like, we looked at the longest common subsequences of transformations (Table 4). Again, we excluded **Cache** and **Macro** transformations from presentation. We again see the preponderance of **Filter**, **Aggregate**, and **Augment** transformations. Beyond that, the most striking feature is the preponderance of **Augment** transformations, particularly in the longer subsequences. To gain more insight into exactly what such sequences of **Augment** transformations are doing, we look more closely at such transformations in the following section.

[5]404 is an HTTP standard response code indicating the requested resource was not found.

6 Cluster Analysis

Recall from Section 5 that three of the most common transformation types in log analysis are **Filter**, **Aggregate** and **Augment**. To find out more details about why and how such transformations are used, we clustered query stages containing these types of transformations, and then examined the distribution of transformations across these clusters. Clustering provides an alternative to manually looking through thousands of examples to find patterns. Similar conclusions would likely have been arrived at using manual coding techniques (i.e., content analysis), but this would have been more time-consuming.

In clustering these transformations, we investigate the following sets of questions:

- What are the different ways in which **Filter**, **Aggregate**, and **Augment** transformations are applied, and how are these different ways distributed?
- Can we identify higher-level tasks and activities by identifying related clusters of transformations? Do these clusters allow us to identify common workflow patterns? What can we infer about the user's information needs from these groups?
- How well do the *commands* in the Splunk query language map to the *tasks* users are trying to perform? What implications do the clusters we find have on data transformation language design?

Length	*Count*	*% Queries*	*Subsequence*
2	2866	16.77	**Transform** | **Aggregate**
2	2675	6.13	**Augment** | **Augment**
2	2446	14.06	**Filter** | **Aggregate**
2	2170	12.70	**Aggregate** | **Rename**
2	1724	8.42	**Filter** | **Augment**
3	2134	12.49	**Transform** | **Aggregate** | **Rename**
3	1430	4.00	**Augment** | **Augment** | **Augment**
3	746	4.24	**Aggregate** | **Augment** | **Filter**
3	718	4.20	**Aggregate** | **Join** | **Filter**
3	717	4.20	**Aggregate** | **Project** | **Filter**
4	710	4.16	**Aggregate** | **Project** | **Filter** | **Rename**
4	710	4.16	**Transform** | **Aggregate** | **Augment** | **Filter**
4	694	2.71	**Augment** | **Augment** | **Augment** | **Augment**
4	472	2.73	**Filter** | **Augment** | **Augment** | **Augment**
4	234	1.37	**Augment** | **Augment** | **Augment** | **Project**
5	280	1.62	**Filter** | **Augment** | **Augment** | **Augment** | **Augment**
5	222	1.30	**Augment** | **Augment** | **Augment** | **Augment** | **Project**
5	200	0.61	**Augment** | **Augment** | **Augment** | **Augment** | **Augment**
5	171	1.00	**Augment** | **Augment** | **Augment** | **Augment** | **Filter**
5	167	0.98	**Filter** | **Augment** | **Augment** | **Augment** | **Aggregate**
6	161	0.94	**Augment** | **Augment** | **Augment** | **Augment** | **Filter** | **Filter**
6	160	0.94	**Augment** | **Augment** | **Filter** | **Filter** | **Filter** | **Augment**
6	160	0.94	**Augment** | **Augment** | **Augment** | **Filter** | **Filter** | **Filter**
6	148	0.87	**Filter** | **Augment** | **Augment** | **Augment** | **Augment** | **Filter**
6	102	0.60	**Augment** | **Aggregate** | **Augment** | **Augment** | **Augment** | **Augment**

Table 4: Longest common subsequences of transformations along with count of how many times such sequences appeared, and the percent of queries they appeared in.

To cluster each set of transformations, we:

(1) parsed each query (see: Section 4)
(2) extracted the stages consisting of the given transformation type,
(3) converted the stages into feature vectors,
(4) projected these feature vectors down to a lower dimensional space using PCA,
(5) projected these features further down into two dimensions, to allow visualization of the clusters, using t-SNE [37], and lastly
(6) manually identified and labeled clusters in the data.

Then, to count the number of transformations in each cluster, we use a random sample of 300 labeled examples from the clustering step to estimate the true proportion of stages in each cluster within 95% confidence intervals. [6]

6.1 Types of Filters

Filter stages primarily consist of the use of the `search` command, which almost all Splunk queries begin with, and which allows users to both select events from a source and filter them in a variety of ways. We clustered all distinct **Filter** stages and discovered 11 cluster types using 26 features[7] (Figure 6). Some of the clusters overlap, in that some examples could belong to more than one group. We discuss how we resolve this below.

The most common application of **Filter** is to use multi-predicate logical conditions to refine an event set, where these predicates are themselves filters of the other types, such as those that look for matches of a given field (e.g., `search status=404`), or those that look for any event containing a specified string (e.g., `search "Authentication failure for user: alspaugh"`). When a **Filter** could go into multiple categories, it was placed into this one, which also contains **Filter**s with many predicates of the same type in a statement with many disjunctions and negations. Thus, it is the largest category. Considering each filter predicate individually might be more informative; we leave that to future work.

Another common **Filter** pulls data from a given source, index, or host (like a `SELECT` clause in SQL). These resemble **Filter**s that look for a match on a given field, but return all events from a given source rather than all events with a specific value in a specific field.

Other types of filters include those that deduplicate events, and those that filter based on time range, index, regular expression match, or the result of a function eval-

[6] Assuming cluster distribution is multinomial with k parameters p_i we use the formula $n = \frac{k^{-1}(1-k^{-1})}{(.05/1.96)^2}$ (which assumes each cluster is equally likely) to estimate the sample size required to estimate the true parameters with a 95% confidence interval. The maximum required size was 246.

[7] See Section 11 for more information about the features used.

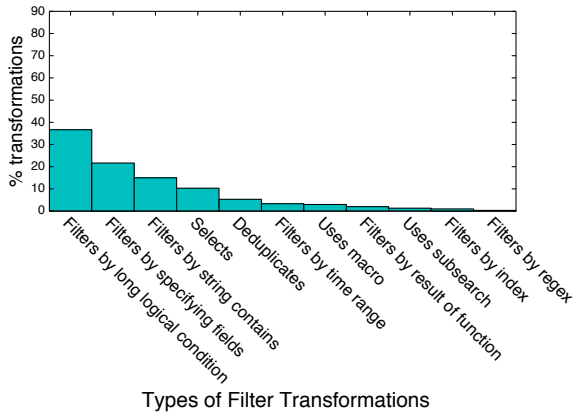


Figure 6: Distribution of different types of **Filter** transformations.

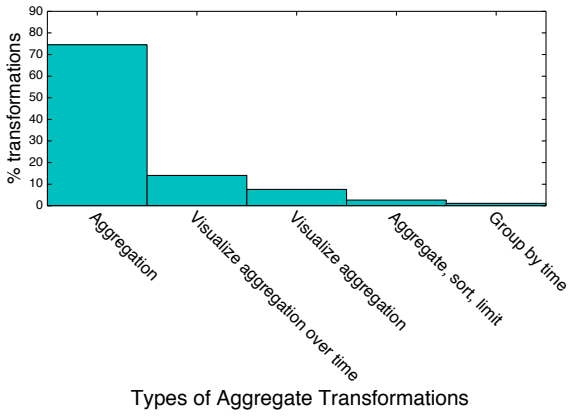


Figure 7: Distribution of different types of **Aggregate** transformations.

uation on the fields of each event. Lastly, some **Filter** transformations include the use of macros, others, the use of subsearches, the results of which are used as arguments to further constrain the current filter.

These use cases reveal several things:

- It is helpful to be able to simultaneously treat log data both as structured (field-value filters, similar to SQL `WHERE` clauses) and as unstructured (string-contains searches, similar to `grep`).
- Some commands in Splunk, like `search`, are heavily overloaded. A redesign of the language could make it easier to identify what users are doing, by bringing the task performed and the command invoked to perform it more in line with one another. For example, there could be a distinct command for each task identified above. This might also form a more intuitive basis on which to organize a data transformation language or interface, but would need to be evaluated for usability.
- Though it may appear that time range searches are not as prevalent as might have be suspected given the importance of the time dimension in log data, this is because the time range is most often encoded in other parameters that are passed along with the query. So time is still one of the most important filter dimensions for log analysis, but this is not reflected in these results.

6.2 Types of Aggregates

We discovered five **Aggregate** cluster types using 46 features (Figure 7). The most common **Aggregate** command is `stats`, which applies a specific aggregation function to any number of fields grouped by any number of other fields and returns the result. Most often, commonplace aggregation functions like `count`, `avg`, and `max` are used. Almost 75% of **Aggregate**s are of this type. Another 10% of **Aggregate**s do this, but then also prepare the output for visualization in a a chart rather than simply return the results (see the "Visualization" tab discussed in Section 3). Another common type of **Aggregate** is similar to these, but first buckets events temporally, aggregates each bucket, and displays the aggregated value over time in a histogram. Another type first aggregates, then sorts, then returns the top *N* results (e.g., `top user`). The last type groups by time, but not necessarily into uniformly sized buckets (e.g., when forming user sessions).

The takeaways from this are:

- Visualizing the results of aggregations is reasonably popular, though much of the time, simply viewing a table of the results suffices. Aggregations lead to the types of relational graphics that many people are familiar with, such as bar and line graphs [36]. Users might also appreciate having the ability to more easily visualize the result of **Filter** transformations as well; for example, using brushing and linking. [8]
- For log analysis, when visualization is used, it is more likely to visualize an aggregate value over buckets of time than aggregated over all time.

6.3 Types of Augments

Augments add or transform a field for each event. The most commonly used such command is `eval`, which is another example of a heavily overloaded command. We discovered eight classes of **Augment** use by clustering over 127 features (Figure 8). These classes shed light onto the results of Section 5 and reveal what some of

[8] Brushing and linking is an interactive visualization technique wherein multiple views of data are linked and data highlighted in one view (i.e., a filter) appears also highlighted in the other view (i.e., a bar graph or heat map).

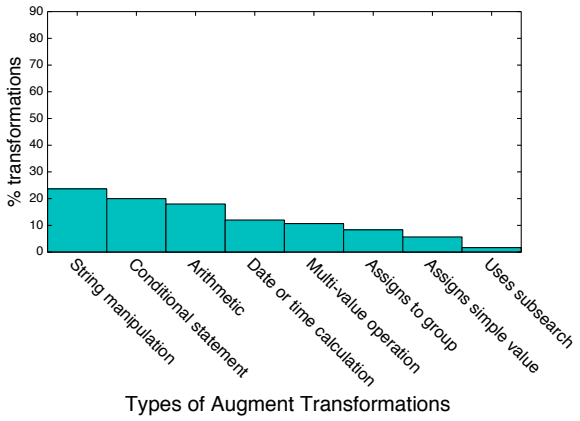


Figure 8: Distribution of different types of **Augment** transformations.

the long pipelines full of **Augment** transformations were likely doing.

The most common ways users transform their data are by manipulating strings (e.g., `eval name=concat(first, " ", last)`), conditionally updating fields (e.g., `eval type=if(status>=400, "failure", "success")`), performing arithmetic (e.g., `eval pct=cnt/total*100`), calculating date-time information (e.g., `eval ago=now()-_time`), applying multi-valued operations (e.g., `eval nitems=mvcount(items)`), or simple value assignments (e.g., `eval thresh=5`). Other `Augment` operations add a field that indicates which group an event belongs to and still others use the results of a subsearch to update a field.

These tasks reveal that:

- Aside from filtering and aggregation, much of log analysis consists of data munging (i.e., translating data from one format into another, such as converting units, and reformatting strings). This is supported by other studies of data analysis in general [28]. Such data munging transformations could be mined to create more intelligent logging infrastructure that outputs data in form already more palatable to end-users, or could be incorporated into an automated system that converts raw logs into nicely structured information. The more complicated transformations should be evaluated to identify whether the tool could be made more expressive.
- Just as with **Filter** transformations, here we observe heavily overloaded commands (i.e., `eval`). Refactoring functionality to clean up the mapping between tasks and commands would help here for the same reasons.

7 Usage Survey

The analytic results open many questions about usage goals that can best be answered by talking to the people who use the system. To this end, we administered a survey to Splunk sales engineers and obtained responses that describe the use cases, data sources, roles, and skill sets of 39 customer organizations. Note: these are not responses directly from customers, rather each sales engineer answered each question once for each of three customers, based on their firsthand knowledge and experience working with those customers. Figure 9 summarizes the results visually.

7.1 Survey Results

The main results are:

User roles: The bulk of Splunk users are in IT and engineering departments, but there is an important emerging class of users in management, marketing, sales, and finance. This may be because more business divisions are interleaving one or more machine generated log data sources for business insights.

Programming experience: Although most Splunk users are technically savvy, most only have limited to moderate amounts of programming experience.

Splunk experience: Surprisingly, many of the customers reported on did not consistently have expertise with Splunk, in fact, some users had no Splunk experience. This may be an artifact of the fact that the survey respondents were sales engineers, who may have opted to reply about more recent or growing customer deployments.

Use cases: Along with the main user roles, the main use cases are also IT-oriented, but, consistent with the other responses, Splunk is sometimes used to analyze business data.

Data sources: Correspondingly, the main type of data explored with Splunk is typical IT data: logs from web servers, firewalls, network devices, and so on. However, customers also used Splunk to explore sales, customer, and manufacturing data.

Transformations applied: Customers primarily use Splunk to extract strings from data, perform simple arithmetic, and manipulate date and time information. In some cases, customers perform more complex operations such as outlier removal and interpolation.

Statistical sophistication: Customers generally do not use Splunk to perform very complicated statistical analysis, limiting themselves to operations like computing descriptive statistics and looking for correlations. In one instance, a customer reported having a team of "math

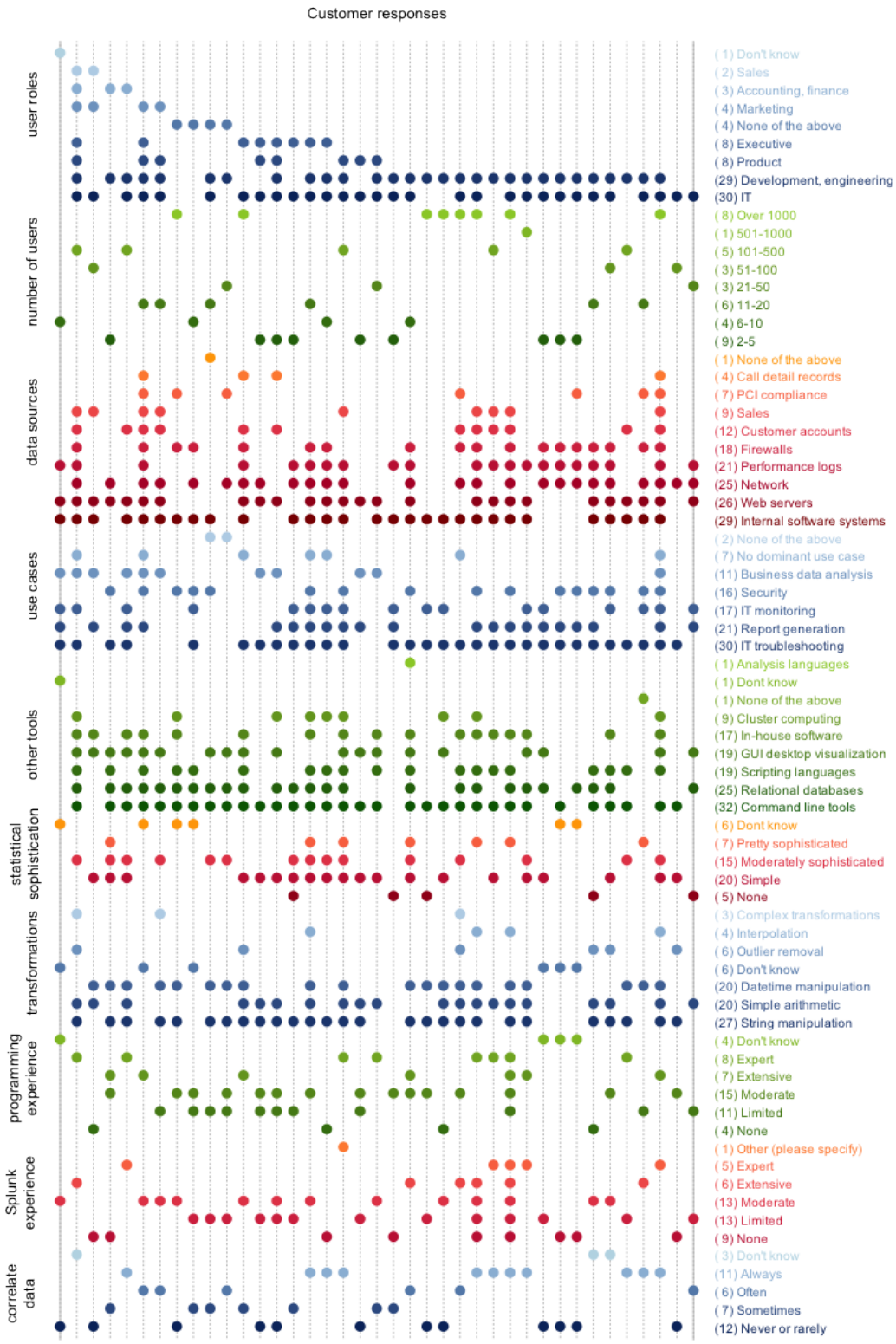


Figure 9: Summary of survey answers. Each vertical line represents a customer. Each colored grouping represents a different question and each row in the group represents one possible response to that question. A dot is present along a given column and row if the option corresponding to that row was selected for the question in that group, for the customer in that column.

junkies" that exported data out of Splunk, ran "very sophisticated batch analytics," and then imported those results back into Splunk for reporting.

Data mash ups: The degree to which customers combine data sources in their analysis varies across individual users and organizations. Some organizations almost always combine data sources for their analysis while a nearly equal number almost never do. This could be in part due to diversity in Splunk expertise and use cases.

Other tools: To better understand the ecosystem in which Splunk exists, we asked what other data analysis tools customers used. In keeping with their IT-oriented roles and use cases, command line tools are frequently used by most Splunk users, in addition to databases, scripting languages, and desktop visualization tools like Tableau. A significant number of customers used custom in-house applications for analyzing their data. A relatively small number used cluster computing frameworks or analysis languages like MATLAB.

Based on these results, we make the following predictions.

- IT and engineering professionals will be increasingly called upon to use their expertise working with machine data to aid other business divisions in their information-seeking needs, and will gain some expertise in these other domains as a result (deduced from user role and use case data).
- Classic tools of the trade for system administrators and engineers will be increasingly picked up by less technical users with other types of training, causing an evolution in both the features offered by the tools of the trade as well as the skills typically held by these other users (deduced from user role data). Although it is likely that more people in a growing variety of professions will learn how to program over the coming years, the market for log and data analysis tools that do not require programming experience will likely grow even faster (deduced from programming experience data).
- There is still no "one stop shop" for data analysis and exploration needs – customers rely on a variety of tools depending on their needs and individual expertise (based on the other tools data). This may be due to the existence of a well-established toolchain where different components are integrated into a holistic approach, not used disparately. Better understanding of which parts of different tools draw users would help both researchers and businesses that make data analysis products understand where to focus their energies.

8 Conclusion

In this paper we presented detailed, quantitative data describing the process of log analysis. While there have been a number of system administrator user studies, there have been few if any quantitative reports on traces of user behavior from an actual log analysis system at the level of detail we provide. In addition, we provide qualitative survey data for high-level context. Together these are important sources of information that can be used to to inform product design, guide user testing, construct statistical user models, and even create smart interfaces that make recommendations to users to enhance their analysis capabilities. We first summarize our main observations, then follow with a call to action for current tool builders and future researchers.

Filtering: In our observations, a large portion of log analysis activity in Splunk consists of filtering. One possible explanation is that log analysis is often used to solve problems that involve hunting down a few particular pieces of data – a handful of abnormal events or a particular record. This could include account troubleshooting, performance debugging, intrusion detection, and other security-related problems. Another possible explanation is that much of the information collected in logs, e.g., for debugging during development, is not useful for end-users of the system. In other words, logs include many different types of data logged for many different reasons, and the difference between signal and noise may depend on perspective.

Reformatting: Our analysis of **Augment** transformations suggested that most of these transformations were for the purpose of data munging, or reformatting and cleaning data. The prevalence of reformatting as a portion of log analysis activity is likely reflective of the fact that much log data is structured in an inconsistent, ad hoc manner. Taken together, the prevalence of filtering and reformatting activity in Splunk suggest that it may be useful for system developers to collaborate with the end users of such systems to ensure that data useful for the day-to-day management of such systems is collected. Alternatively, another possible explanation is that the Splunk interface is not as useful for other types of analysis. However, other reports indicate that indeed, much of data analysis in general does involve a lot of data munging [28].

Summarization: We observed that it is common in Splunk to **Aggregate** log data, which is a way of summarizing it. Summarization is a frequently-used technique in data analysis in general, and is used to create some of the more common graph types, such as bar charts and line graphs [36]. This suggests it may be useful to automatically create certain types of summarization

to present to the user to save time. In log analysis with Splunk, summarizing with respect to the time dimension is an important use case.

The complexity of log analysis activity: We were not able to determine whether Splunk users make use of some of the more advanced data mining techniques proposed in the literature, such as techniques for event clustering and failure diagnosis [20, 25]. One possible explanation for this is that due to the complexity and variability of real world problems, as well as of logged information, designing one-size-fits-all tools for these situations is not feasible. Alternatively, such analyses may occur using custom code outside of Splunk or other analytics products as part of a large toolchain, into which we have little visibility. This idea is supported by some of the Splunk survey results (Section 7). Other possible explanations include lack of problems that require complicated solutions, lack of expertise, or requirements that solutions be transparent, which may not be the case for statistical techniques. It could also be the case that such techniques are used, but are drowned out by the volume of data munging activity. Finally, it may be that we simply were not able to find more complex analytics pipelines because programmatically identifying such higher-level activities from sequences of smaller, lower-level steps is a difficult problem.

Log analysis outside of IT departments: Our survey results also suggest that log analysis is not just for IT managers any longer; increasing numbers of non-technical users need to extract business insights from logs to drive their decision making.

9 Future Work

Need for integrated provenance collection: Understandably, most data that is logged is done so for the purpose of debugging systems, not building detailed models of user behavior [3]. This means that much of the contextual information that is highly relevant to understanding user behavior is not easily available, and even basic information must be inferred through elaborate or unreliable means [10]. We hope to draw attention to this issue to encourage solutions to this problem.

Improving transformation representation: In the process of analyzing the query data, we encountered difficulties relating to the fact that many commands in the Splunk language are heavily overloaded and can do many different things. For example, `stats` can both aggregate and rename data. When this is the case, we are more likely to have to rely on error-prone data mining techniques like clustering and classification to resolve ambiguities involved in automatically labeling user activities. If the mapping between analysis tasks and analysis representation (i.e., the analysis language) were less muddied, it would alleviate some of the difficulties of analyzing this activity data and pave the way for easier modeling of user behavior.

Opportunities for predictive interfaces: Thinking forward, detailed data on user behavior can be fed into advanced statistical models that return predictions about user actions. Studies such as the one we present are important for designing such models, including identifying what variables to model and the possible values they can take on. Other important variables to model could include who the user is, where their data came from, and what problems they are trying to solve. These could be used to provide suggestions to the user about what they might like to try, similar to how other recently successful tools operate [27].

Further analyses of data analysis activity: Finally, in this paper, we only presented data analysis activity from one system. It would be informative to compare this to data analysis activity from other systems, and on other types of data besides log data. Thus, we make our analysis code public so others can more easily adapt and apply our analysis to more data sets and compare results.

10 Acknowledgments

Thanks to David Carasso for being the go-to person for answering obscure questions about Splunk. Thanks to Ari Rabkin and Yanpei Chen for feedback on early drafts. This research is supported in part by NSF CISE Expeditions Award CCF-1139158, LBNL Award 7076018, and DARPA XData Award FA8750-12-2-0331, and gifts from Amazon Web Services, Google, SAP, The Thomas and Stacey Siebel Foundation, Adobe, Apple, Inc., Bosch, C3Energy, Cisco, Cloudera, EMC, Ericsson, Facebook, GameOnTalis, Guavus, HP, Huawei, Intel, Microsoft, NetApp, Pivotal, Splunk, Virdata, VMware, and Yahoo!.

11 Availability

The code used to generate the facts and figures presented in this paper, along with additional data that was not included due to lack of space, can be found at:

`https://github.com/salspaugh/lupe`

References

[1] ALSPAUGH, S., ET AL. Towards a data analysis recommendation system. In *OSDI Workshop on Managing Systems Automatically*

and Dynamically (MAD) (2012).

[2] ALSPAUGH, S., ET AL. Building blocks for exploratory data analysis tools. In *KDD Workshop on Interactive Data Exploration and Analytics (IDEA)* (2013).

[3] ALSPAUGH, S., ET AL. Better logging to improve interactive data analysis tools. In *KDD Workshop on Interactive Data Exploration and Analytics (IDEA)* (2014).

[4] BARRETT, R., ET AL. Field studies of computer system administrators: Analysis of system management tools and practices. In *ACM Conference on Computer Supported Cooperative Work (CSCW)* (2004).

[5] BITINCKA, L., ET AL. Optimizing data analysis with a semi-structured time series database. In *OSDI Workshop on Managing Systems via Log Analysis and Machine Learning Techniques (SLAML)* (2010).

[6] CHEN, Y., ET AL. Design implications for enterprise storage systems via multi-dimensional trace analysis. In *ACM Symposium on Operating Systems Principles (SOSP)* (2011).

[7] CHEN, Y., ET AL. Interactive analytical processing in big data systems: A cross-industry study of mapreduce workloads. *International Conference on Very Large Databases (VLDB)* (2012).

[8] CHIARINI, M. Provenance for system troubleshooting. In *USENIX Conference on System Administration (LISA)* (2011).

[9] COUCH, A. L. "standard deviations" of the "average" system administrator. USENIX Conference on System Administration (LISA), 2008.

[10] GOTZ, D., ET AL. Characterizing users' visual analytic activity for insight provenance. *IEEE Information Visualization Conference (InfoVis)* (2009).

[11] GRAY, J. Why do computers stop and what can be done about it? Tech. rep., 1985.

[12] HAUGERUD, H., AND STRAUMSNES, S. Simulation of user-driven computer behaviour. In *USENIX Conference on System Administration (LISA)* (2001).

[13] HREBEC, D. G., AND STIBER, M. A survey of system administrator mental models and situation awareness. In *ACM Conference on Computer Personnel Research (SIGCPR)* (2001).

[14] JANSEN, B. J., ET AL. Real life information retrieval: A study of user queries on the web. *SIGIR Forum* (1998).

[15] KLEIN, D. V. A forensic analysis of a distributed two-stage web-based spam attack. In *USENIX Conference on System Administration (LISA)* (2006).

[16] LAENDER, A. H. F., RIBEIRO-NETO, B. A., DA SILVA, A. S., AND TEIXEIRA, J. S. A brief survey of web data extraction tools. *SIGMOD Record* (2002).

[17] LAO, N., ET AL. Combining high level symptom descriptions and low level state information for configuration fault diagnosis. In *USENIX Conference on System Administration (LISA)* (2004).

[18] LOU, J.-G., ET AL. Mining dependency in distributed systems through unstructured logs analysis. *SIGOPS Operating System Review* (2010).

[19] LOU, J.-G., FU, Q., WANG, Y., AND LI, J. Mining dependency in distributed systems through unstructured logs analysis. *SIGOPS Operating System Review* (2010).

[20] MAKANJU, A. A., ET AL. Clustering event logs using iterative partitioning. In *ACM International Conference on Knowledge Discovery and Data Mining (KDD)* (2009).

[21] MARMORSTEIN, R., AND KEARNS, P. Firewall analysis with policy-based host classification. In *USENIX Conference on System Administration (LISA)* (2006).

[22] NORIAKI, K., ET AL. Semantic log analysis based on a user query behavior model. In *IEEE International Conference on Data Mining (ICDM)* (2003).

[23] OLINER, A., ET AL. Advances and challenges in log analysis. *ACM Queue* (2011).

[24] OLINER, A., AND STEARLEY, J. What supercomputers say: A study of five system logs. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2007).

[25] OLINER, A. J., ET AL. Using correlated surprise to infer shared influence. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2010).

[26] OLSTON, C., ET AL. Pig latin: A not-so-foreign language for data processing. In *ACM International Conference on Management of Data (SIGMOD)* (2008).

[27] OTHERS, S. K. Wrangler: Interactive visual specification of data transformation scripts. In *ACM Conference on Human Factors in Computing Systems (CHI)* (2011).

[28] OTHERS, S. K. Enterprise data analysis and visualization: An interview study. In *Visual Analytics Science & Technology (VAST)* (2012).

[29] PANG, R., ET AL. Characteristics of internet background radiation. In *ACM SIGCOMM Internet Measurement Conference (IMC)* (2004).

[30] PIKE, R., ET AL. Interpreting the data: Parallel analysis with sawzall. *Scientific Programming Journal* (2005).

[31] PINHEIRO, E., ET AL. Failure trends in a large disk drive population. In *USENIX Conference on File and Storage Technologies (FAST)* (2007).

[32] POBLETE, B., AND BAEZA-YATES, R. Query-sets: Using implicit feedback and query patterns to organize web documents. In *International Conference on World Wide Web (WWW)* (2008).

[33] RICHARDSON, M. Learning about the world through long-term query logs. *ACM Transactions on the Web (TWEB)* (2008).

[34] SIGELMAN, B. H., AND OTHERS. Dapper, a large-scale distributed systems tracing infrastructure. Tech. rep., Google, Inc., 2010.

[35] SILVESTRI, F. Mining query logs: Turning search usage data into knowledge. *Foundations and Trends in Information Retrieval* (2010).

[36] TUFTE, E., AND HOWARD, G. *The Visual Display of Quantitative Information*. Graphics Press, 1983.

[37] VAN DER MAATEN, L., AND HINTON, G. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research (JMLR)* (2008).

[38] VELASQUEZ, N. F., ET AL. Designing tools for system administrators: An empirical test of the integrated user satisfaction model. In *USENIX Conference on System Administration (LISA)* (2008).

[39] XU, W., ET AL. Experience mining google's production console logs. In *OSDI Workshop on Managing Systems via Log Analysis and Machine Learning Techniques (SLAML)* (2010).