Web Analytics and the Art of Data Summarization

Archana Ganapathi and Steve Zhang
Splunk Inc.
{aganapathi, steveyz}@splunk.com

Abstract

Web Analytics has become a critical component of many business decisions. With an ever growing number of transactions happening through web interfaces, the ability to understand and introspect web site activity is critical. In this paper, we describe the importance and intricacies of summarization for analytics and report generation on web log data. We specifically elaborate on how summarization is exposed in Splunk and discuss analytics search design trade-offs.

1 Introduction

Modern websites contain a wealth of content to provide easy and efficient access to information about an enterprise. These websites range from simple static html to very sophisticated dynamic content. While hosting such content comes with its own set of challenges, such as resource provisioning and traffic anomaly detection, an even larger challenge is to identify business insights based on web access patterns. The most useful source of such insight is the web access log. Mining these logs can provide information on *what happened*, *what to anticipate* and *how well things are working*.

Many enterprises rely on web analytics for business intelligence, i.e., to evaluate and optimize their business decisions. Mining logs can help answer questions such as *what search terms are effective in directing traffic to our website* and *how different are our visitor demographics now compared to last quarter*. Such information can help inform content layout and search engine ad word campaigns. Furthermore, conversion funnels created with such data, when overlaid with revenue metrics, can inform strategic initiatives for an enterprise.

Web data, although immensely informative, can often contain millions of events per second based on traffic volume and verbosity of logging. For example, logging every click on a page drastically increases the number of events generated compared to logging a single event per pageview. Much of web analytics tries to extract useful patterns and statistics periodically to generate regular reports. Thus, although the data itself changes frequently, the report-generating searches seldom change.

With high volumes of data, it can take hours to generate reports across data spanning large time windows. If daily reports take hours to execute, very little time is left to investigate anomalies and make changes in time for the next roll out. Furthermore, many investigations benefit from correlating web log data with alternate data sources, such as application logs, system performance metric logs or other databases. Performing such correlations via ad hoc searches can be prohibitively expensive if it requires rerunning the report's search to account for trends indicated by an additional data source. While sampling large data volumes helps reduce the quantity of data to search, it does very little to empower business units with a complete perspective of trends and metrics.

We propose using data summarization to efficiently search large quantities of log data. Data summarization involves running a search at regular intervals to extract information from the raw data. The results of this search are stored in a *summary index*, against which subsequent searches and reports can be run. Since the summaries of the data are smaller, these searches execute much faster than those run against the raw data. Furthermore, if many different reports have common search elements, the common subsearches can be summarized to make

SLAML '11, October 23, 2011, Cascais, Portugal.

multiple reports run efficiently. Even for the same report, it is more efficient to run the report against varying time granularities. For example, a daily summary of the data would significantly speed up a weekly report on the data as well as a monthly report on the data. As a result, the computational cost of running a search over large volumes of data is amortized over time by running the search periodically on much smaller quantities of data.

In this paper, we examine data summarization applied to the specific problem of log-based web analytics. We store and perform analytics on web access logs using Splunk, a platform to index and search semi-structured time series data. We explain our use of Splunk to add summaries on top of our web logs, and subsequently run report-generating searches for web analytics.

The remainder of this paper resumes by comparing various approaches to web analytics in Section 2 and details what information web logs contain in Section 3. In Section 4, we explain in depth how data summarization is implemented in Splunk and provide detailed examples of searches, comparing their performance against summarized and unsummarized data. We also discuss various challenges involved when searching summarized data instead of raw data. Section 5 provides a brief overview of related work and Section 6 concludes.

2 Web Analytics Background

There are many well known commercial products for performing web analytics. Rather than delve into detailed feature lists for these product we thought it best to focus on guiding principles for what data is used, how much of it is stored and implications at retrieval time.

The first and foremost consideration is what data to use for web analytics. Traditionally there have been two schools of thought on what web data is used - i) javascript tags per page, and ii) web access logs. The largest difference between these two data collection methods is the degree of invasiveness of the approaches. Tools such as Omniture [7] and Google Analytics [3] require instrumenting each web page with javascript code snippets and mining events returned by the beacons for analytics. While this approach allows fine-grained instrumentation of web pages, the downside is that instrumentation can be invasive and cause significant slowdowns in page load times. Furthermore, if an organization has strict cyclical website releases, any analysis that requires additional/new instrumentation must wait until the next release cycle. Analytics that solely relies on javascript beaconing also suffers from insufficient information to perform historical trend analysis, especially if the website, or even pages, have evolved over time. The second school of thought solely relies on web server logs for analytics. Google Urchin [4] is an example of a tool that relies on web server log analysis. All pages are treated equally, and no instrumentation is required for each page addition or modification. While a significant advantage of this approach is that it is easier to perform analytics as far back as the logs are available, the challenge is in handling changes in log format over time. Furthermore, with dynamically generated web pages, various key-value pair url parameters must be understood and mined although their churn rate is high.

The ideal web analytics approach is to overlay log data with custom javascript beacons to allow both fine-grained instrumentation as well as historical analysis of web access patterns. Additionally, any web analytics system must adapt and account for changes over time and not require re-indexing historical data to account for new *schema*.

The second consideration for web analytics is how much and what granularity of information is stored. There are three options when considering what to store - (i) all the raw data, (ii) samples of the data, or (iii) summaries of the data. Ideally, one could store and search all the raw data across all time. However, many web analytics products have hard bounds on how much data can be stored at a time, or require a preprocessing step to impose a specific schema to the data. The advantage of sampling the data is that you can reduce the quantity of data stored. In fact, Google Analytics imposes data sampling when the traffic volume exceeds a certain threshold. The major downside of data sampling is that spikes and anomalies are often overlooked and difficult to detect, especially with a coarse sampling granularity. The third option, data summarization, addresses this issue by maintaining summary statistics for various characteristics of the data rather than the raw data itself, and is used by commercial tools such as Omniture. Summarizing average traffic volume on a daily basis provides a significant space saving compared to maintaining all the raw data if the only goal is to calculate traffic volume. That said, although problem *detection* is easier with summarization than sampling, problem diagnosis is still difficult without the raw unsummarized data. Furthermore, if the summaries are over a coarse enough time granularity, depending on the metrics maintained (e.g. average but not max), spikes may be smoothed out. This shortcoming can only be addressed by preserving the original data and allowing drilldowns from the summaries as and when necessary.

The third and often most impactful consideration is how easy it is to retrieve the data and search or run reports against it. There is a wide range of possibilities for data retrieval that span from limited predetermined metrics or reports (e.g. Google Analytics) all the way to needle-in-a-haystack type fine-grained plain text searches. With business needs evolving over time, it is

difficult to anticipate what metrics to track a priori and many times, the need to overlay multiple datasets with web access data greatly influences the decision. Since time is often the only axis with which multiple datasets can be interleaved, it is important to preserve temporal patterns in as fine granularity as possible.

By nature of implementation, as described in [8], Splunk handles unstructured and semi-structured data efficiently, and does not require specifying fields to extract up front. We scope the remainder of this paper to web access log analysis and use Splunk to summarize the data while preserving drilldown capability to the raw data. We also explain how to leverage Splunk for data summarization in addition to searching and reporting on web logs. In the next section, we describe typical information in web logs and useful ways to leverage the information.

3 What can my web logs tell me?

Web access logs contain a wealth of information about traffic served by web servers. There are a variety of web servers, such as Apache [2], IIS [5] and Nginx [6]. Each of these web servers supports a variety of formats for web logs generated. For example, Apache produces both the access combined and access common formats, each with a variety of customizable fields. Although the range of possible web log formats is wide, there are several required fields captured by all these formats. Figure 1 shows a sample snapshot from an Apache log file. Two categories of information can be extracted from the logs:

1. **Visitor demographics:** Fields such as clientip tell you the visitor's ip address, which can be used to determine geolocation of visitors. Fields such as useragent can indicate the browser or platform used by site visitors.

2. **Visitor activity and site usage patterns:** Fields such as uri_path show what pages on your site are visited. URL parameters can be used to determine what content, if any, was downloaded. Additional analytics on uri fields can show information on how many pages people visit, in what order etc.

At a cursory level, this information helps the web operations team better provision and manage resources to adapt for load and popularity. However, a few transformations to the data can lead to business-level insights such as marketing strategy, product positioning and revenue channels. There are two transformations we identified as crucial to improving business insights:

1. **Sessionize the data:** Use clientip, useragent and/or cookie as well as any temporal thresholds necessary to define what entails a usage session. Metrics calculated on sessions make it easier to characterize user behavior and therefore expose content based on common actions that lead to a conversion.

2. **Coalesce external data sources:** Often, web logs alone do not tell you conclusive information, but when interleaved with another data source, such as a revenue database, can produce more concrete metrics. For instance, web logs can show what search terms users searched for that caused search engines to refer them to your website. While a count of visits by search term produces a popularity metric, it is more meaningful when you multiply this metric with the amount spent on ad words for the corresponding search phrases and keywords. Correlating the popularity data with a marketing campaign cost sheet will allow us to evaluate the return on investment of an ad campaign.

Splunk is an appropriate choice for performing such analytics as its search language allows users to express complicated analytics tasks effectively. Also, a major advantage of Splunk is that it allows us to coalesce multiple data sources using either or both temporal and non-temporal characteristics, making it much more feasible to perform ad hoc introspection and analytics across multiple data sources. Figure 2 shows an example of a web analytics dashboard built using Splunk searches.

We next explain how data summarization is implemented in Splunk and discuss the intricacies of searching the summary data.

4 Data Summarization in Splunk

Data summarization is crucial to sift through large volume of historical data in a timely fashion. Most approaches to Web Analytics, and Business Intelligence in general, involve acting upon already summarized data. Splunk leverages a MapReduce-like architecture on top of its indexed data store to allow users to search and perform analytics on large quantities of data. For details on Splunk's architecture, we refer the reader to [8]. Since Splunk is designed to effectively handle large volumes of raw textual data, Splunk treats summary data much like it treats raw data. Splunk queries can compute summaries and render the output as text, allowing Splunk to index and store that output in the same way it consumes the original log data.

4.1 Summary Data Layout

Since Splunk has full control of how summary data is rendered as text, we naturally chose a format that is the easiest and most efficient for the system to process. Although Splunk can interpret most common log formats

```
96.61.62.162 - - [10/Aug/2011:09:58:03 -0700] "POST /page/sign_up/download HTTP/1.1" 302 20 "https://www.splunk.com/index.php/sign_up/download" "Mozilla/5.0
(Windows NT 6.1; WOW64) AppleWebKit/535.1 (KHTML, like Gecko) Chrome/14.0.814.0 Safari/535.1" "96.61.62.162.1312995221070430"
157.55.116.33 - - [10/Aug/2011:09:58:07 -0700] "GET /wiki/index.php?title=Special:RecentChanges&hidemyself=0&days=30&hideliu=1&from=20101221055615&feed=atom
HTTP/1.1" 200 13004 "-" "msnbot/2.0b (+http://search.msn.com/msnbot.htm)._" "157.55.116.33.1312995487208430"
96.61.62.162 - - [10/Aug/2011:09:58:07 -0700] "GET /download HTTP/1.1" 302 199 "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.1 (KHTML, like Gecko)
Chrome/14.0.814.0 Safari/535.1" "96.61.62.162.1312995221070430"
```

Figure 1: Example Apache web log lines for www.splunk.com

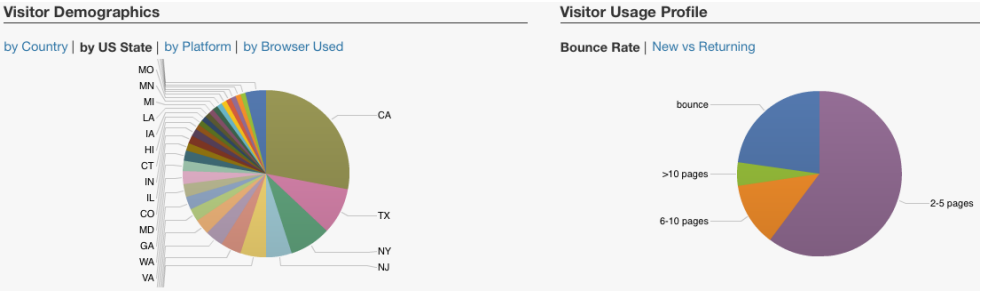

Figure 2: Example of dashboards built using Splunk searches against web log data.

series	kb
audittrail	3230.477770
scheduler	2472.734764
splunkd	20566.588819
splund_access	81.030276

Table 1: Table rendering of summary data

automatically (and with simple configuration changes handle the rest of the long tail of textual data formats), not all formats are equally easy to interpret. Similar to other big-data technologies that rely on a divide-and-conquer strategy for scalability (e.g. Hadoop), Splunk prefers data that is denormalized. Splunk also automatically extracts field value pairs separated by an equals sign. As a result, summary data is stored in a format that is easy for Splunk to interpret.

Figure 3 shows example summary text, rendered from a report over a single day of the total KB of data indexed by data type. Note that each row of the report is rendered as a separate event, and properties of the search itself (e.g. the search time range) are added to each event. This denormalized format allows each summary row to be retrieved and used independently. Furthermore, it allows the process of reducing summaries to get the final report to be easily parallelized. Such parallelization is essential for efficiently handling *big data*. The report is normally rendered in the Splunk UI as a table similar to Table 1.

4.2 Searching Summarized Data

Splunk treats summary data the same as raw log data. Essentially, whether a piece of data represents a summary or a raw event log is opaque to Splunk. This is advantageous for several reasons. First, because time is treated as a first-class concept within Splunk, retrieving summaries for a specific timerange is very efficient. Second, since everything is indexed, we can quickly retrieve specific summary entries, such as those that pertain to a given URI or username, without having to retrieve all summary entries that may have been generated with those entries. Furthermore, a multitude of options for configuring and managing the lifecycles and access controls of raw data are made available for summary data. For example, older data can be moved to a slower, cheaper storage option after it reaches a certain age or once some amount of disk space is used. This consistency in treatment of raw and summary data does present some additional challenges. Mainly, a user of the system must specify exactly how the summary data is to be generated and how it can be used to produce a final report. However, this apparent disadvantange also presents a silver lining for advanced users, who are given ultimate control in how summaries are generated and reduced. This flexibility is especially important when handling data that does not summarize trivially, such as computing distinct counts for a high cardinality field. The next section describes common summarization pitfalls.

4.3 Summarization Challenges/Trade-offs

Although the common case of summarizing data with sufficient statistics over a time period is fairly straightforward, there are exceptional cases where the summarization leads to an incomplete or incorrect view of the data. Below, we discuss four such issues in the context of web analytics.

The cardinality curse:

Picturing the data in a table format, summarization

```
06/13/2011 00:00:00, info_min_time=1307948400.000, info_max_time=1308034800.000, info_search_time=1308089612.499, kb="3230.477770", series=audittrail
06/13/2011 00:00:00, info_min_time=1307948400.000, info_max_time=1308034800.000, info_search_time=1308089612.499, kb="2472.734764", series=scheduler
06/13/2011 00:00:00, info_min_time=1307948400.000, info_max_time=1308034800.000, info_search_time=1308089612.499, kb="20566.588819", series=splunkd
06/13/2011 00:00:00, info_min_time=1307948400.000, info_max_time=1308034800.000, info_search_time=1308089612.499, kb="81.030276", series=splunkd_access
```

Figure 3: Example Splunk summary data

helps reduce the number of *rows* in the table. However, based on the dimensionality of various *columns* in the table, the number of rows removed by summarization may not be substantial. For example, an hourly summary containing the number of hits by http status code compresses the data into 10s of rows per hour (i.e. at most one per status code). However, a count by clientip does little to reduce the data as the number of possible clientips is large compared to the number of status codes.

The impact of summarization is most visible when searching the summaries to compute statistics. With each additional data dimension on which statistics are calculated, there is a combinatorial increase in the cardinality of data searched. Thus, it is important to be judicious when deciding how to summarize the data. A rule of thumb is to maintain separate summaries for metrics across unrelated *columns* rather than a single summary with data pivoting multiple *columns*.

Border patrol:

Summarizing over regular intervals is practical for most metrics. For example, calculating and storing total hits on an hourly basis allows you to quickly aggregate hits on a daily basis by summing over the hourly totals. However, summarizing sessionized data can introduce inaccuracies when computing statistics. For instance, a user session that starts at 10:55am and ends at 11:05am would be incorrectly summarized by the hourly summaries at 11am and at noon. Two solutions to clean up these summaries are (i) only summarize sessions after the sessions have definitively ended, or (ii) periodically make a pass over the summarized sessions to perform any clean-up to merge sessions straddling multiple summary windows. The downside of option (i) is that searches to populate the summary index would have to span more than the last hour of data, which could prove expensive. The downside of option (ii) is that any automation to periodically perform cleanup will have to propagate the clean up to searches/summaries relying on the summarized data. The cost of such cleanups far outweighs the disadvantages of double-counting a handful of "border-straddling" sessions.

Caveats of statistics:

While the idea of searching summaries instead of searching the raw data is appealing and often more efficient, we thought it important to provide some words of caution. Typically, summaries are used to speed up the computation of averages and counts over time. There are several pitfalls the user must avoid when performing statistics using summarized data. For example, to compute averages in a final report, users must be aware that their summaries must include sums and counts rather than averages. They must also be explicit in the query about how the sums and counts are to be combined to compute the averages. Splunk mitigates this problem by adding commands to its query language that simplify many common cases.

One should avoid storing averages and distinct counts as summary fields to avoid potential miscalculation of these metrics at search time. For instance, to calculate average number of hits per visitor over 5 hours using hourly summaries, storing averages per hour and calculating an average over the 5 hourly summaries would produce the wrong result. Instead, it is more accurate to store the total number of hits per visitor as a multivalue list and then compute the average at search time. Splunk exposes some commands, as discussed in Section 4.2, to simplify several common case statistics that are frequently summarized, such as average and top. However, it is important to carefully balance trade-offs in the more complicated scenarios such as calculating distinct counts, where the user has many options for applying approximation heuristics to trade accuracy for efficiency.

Resurrecting pre-summarized data:

An important issue for log analysis is exposing appropriate drilldown behavior from summarized data. For example, if a particular time period shows a spike in traffic, we must drilldown to the original unsummarized data to search for potential triggers or culprits causing the spike. However, if we have multi-level summaries, it may be expensive for the drilldown to search the entire set of events coerced into the summary. That said, if the original pre-summarized data is discarded, for instance to save disk space, there is no perceivable method to *zoom in* further for understanding what caused the spike.

With these caveats in mind, we next show examples of web analytics searches and evaluate their performance against summarized and unsummarized data in Splunk.

4.4 Summarization Effectiveness

In this section we provide examples of composing Splunk searches against raw web logs as well as summarized versions of the log data. We compare the performance of searches towards summarized and unsummarized data and explain trade-offs between the approaches.

4.4.1 User session summarization search

One of the most insightful operations on web log data is to sessionize events and perform analytics on sessions. To this extent, we explain how to construct a user session in Splunk. We subsequently examine the performance of running searches against summarized and unsummarized user session data. Splunk automatically extracts fields for Apache access common and access combined log formats as well as IIS formats; for the purpose of this example, we examine our Apache web access logs.

A user session can be defined using the search below:

$[get\_user\_sessions]$
$source = my\_access\_combined\_log\ status = 200$
$\quad | \ transaction\ clientip\ maxpause = 1h$

The first part of the search specifies a path to the file the events must be from and further constrains it to events with status=200. Apache events with status=200 are then piped to the next stage of the search, which uses the *transaction* command to sessionize events with the same clientip such that there is no more than an hour gap between adjacent events in a session. Note that the transaction command also supports specifying a maxspan parameter to specify the maximum window size of a session, and startswith and endswith parameters to use non-temporal boundaries for a sessions.

To demonstrate searching against sessionized data and understand the performance advantages of summarization, we focus here on a search to breakdown the number of pages visited during a user session. The search run against the raw data is as follows:

$`get\_user\_sessions` \ | \ eval\ user\_type =$
$\quad case(eventcount = 1,\ \text{“}bounce\text{”},$
$\quad\quad eventcount <= 5,\ \text{“}2 - 5\ pages\text{”},$
$\quad\quad eventcount <= 10,\ \text{“}6 - 10\ pages\text{”},$
$\quad\quad eventcount > 10,\ \text{“} > 10\ pages\text{”})$
$\quad | \ stats\ count\ by\ user\_type$

The sessionization search is saved as a macro called *get_user_session* and used in the above search. The sessionized events are piped to add a user_type column that buckets users into 4 categories based on the eventcount per session. Finally, we calculate a count per user_type.

To populate a summary index with user session data, we run the search below:

$`get\_user\_sessions` \ | \ stats\ list(uri\_path)\ as\ uri\_path,$
$\quad max(duration)\ as\ myduration,$
$\quad max(eventcount)\ as\ myeventcount,$
$\quad values(eventtype)\ as\ myeventtypes,$
$\quad min(req\_time)\ as\ earliesttime,$
$\quad max(req\_time)\ as\ latesttime\ by\ clientip,\ \_time$

This search extracts the necessary and sufficient statistics from the user sessions and stores them in a summary index so various searches can leverage this information. For the purpose of this experiment, we configured this search to run on an hourly schedule, searching and summarizing the previous hour of data and storing it in a summary index we named summary_hourly. To run our search to calculate pages per session based on the summarized data, we simply replace the ‘get_user_session‘ macro with our summary index name as follows:

$index = summary\_hourly$
$\quad | \ eval\ eventcount = myeventcount$
$\quad | \ eval\ user\_type =$
$\quad case(eventcount = 1,\ \text{“}bounce\text{”},$
$\quad\quad eventcount <= 5,\ \text{“}2 - 5pages\text{”},$
$\quad\quad eventcount <= 10,\ \text{“}6 - 10pages\text{”},$
$\quad\quad eventcount > 10,\ \text{“} > 10pages\text{”})$
$\quad | \ stats\ count\ by\ user\_type$

Furthermore, we can populate a daily summary index based on scheduled searches towards the hourly summary index as follows:

$index = summary\_hourly$
$\quad | \ transaction\ clientip\ maxpause = 1h$
$\quad | \ stats\ list(uri\_path)\ as\ uri\_path,$
$\quad\quad sum(myduration)\ as\ myduration,$
$\quad\quad sum(myeventcount)\ as\ myeventcount,$
$\quad\quad values(myeventtypes)\ as\ myeventtypes,$
$\quad\quad min(earliesttime)\ as\ earliesttime,$
$\quad\quad max(latesttime)\ as\ latesttime\ by\ clientip,\ \_time$

Here, we re-transact on the data from the hourly summary to handle the border conditions where a session straddles an hour boundary. The subsequent search against the daily summary is the same as the search against the hourly summary, except with summary_hourly replaced by summary_daily.

The summary index populating searches are scheduled to run periodically and any searches against these summaries can be run in an ad hoc manner. To demonstrate the performance implications of summarization, we ran the page visits per session search against the unsummarized data, against the hourly summaries and against daily summaries for www.splunk.com web logs spanning January 1, 2011 to May 31, 2001. The results are summarized in table 2. There is an 80% improvement in search runtime when we search against the hourly summary compared to the raw unsummarized data. However, there is only a 5% improvement when using the

summary granularity	time (hr:min:sec)
unsummarized	01:38:40
hourly summary	00:08:11
daily summary	00:07:47

Table 2: Comparison of search performance for calculating pages visited per session against unsummarized data, hourly summaries, and daily summaries

summary granularity	time (hr:min:sec)
unsummarized	01:36:46
hourly summary	00:16:00
daily summary	00:15:49

Table 3: Comparison of search performance for calculating top landing pages against unsummarized data, hourly summaries, and daily summaries

daily summary over the hourly summary. Here, very few sessions compress down in the daily summaries, likely because we define maxpause to be one hour.

We also evaluate the performance of a second search, that calculates the top landing page per session, towards the same summary indexes. The landing page search against the raw data is constructed as follows:

$$\begin{aligned}&`get\_user\_sessions` \mid search\ eventcount > 1\\ &\quad \mid eval\ landing\_page = mvindex(uri\_path,\ 0)\\ &\quad \mid fields\ landing\_page \mid top\ landing\_page\end{aligned}$$

Sessionizing the data by clientip produces a list of uris visited during the session in the uri_path field. The segment that calculates the landing_page uses the mvindex command, which allows you to access a specific element of a multivalue list. The same search against the hourly summary index is expressed as:

$$\begin{aligned}&index = summary\_hourly\\ &\quad \mid makemv\ delim = \text{“ ”}\ uri\_path\\ &\quad \mid search\ eventcount > 1\\ &\quad \mid eval\ landing\_page = mvindex(uri\_path,\ 0)\\ &\quad \mid top\ landing\_page\end{aligned}$$

The addition of the search segment with the makemv command is meant to ensure that the uri_path field is treated as a multivalue field where each element in the uri_path list can be accessed separately. Note that these searches can be used to calculate top exit page by using mvindex(uri_path,-1) to identify the last element visited in the uri sequence.

Table 3 compares the performance of running the top landing page search against unsummarized data, the hourly summary and the daily summary. Similar to the page visits per session search, we see an order or magnitude improvement in performance when using the hourly summary instead of the raw unsummarized data. Again, there is little gain in using daily summaries over hourly summaries since per-user-session data does not compress well on a daily granularity.

We can optimize what we summarize and store by limiting the summary statistics to a single search. For example, to maintain a summary index for calculating page visits per session, we only need to store a count by user_type in the hourly and daily summaries. Similarly, a separate summary for count by landing page can be used to feed a top landing page search. Of course, the drawback of such an optimization would be the space overhead of an additional summary as well as the lack of reusability of the summary for other user-session-specific searches.

4.4.2 Web traffic by uri and status

In the next example, we examine the advantage of constraining a summary to cater to a specific search rather than an exhaustive set. Specifically, we focus on a search to break down web traffic by url and status code and compare the performance of using a single summary for both good and bad status codes versus separate summaries for the two categories. We populate our hourly summary index for web traffic as follows:

$$\begin{aligned}&eventtype = web - traffic - external\\ &\quad \mid stats\ count\ as\ \text{“}hits\text{”},\\ &\quad\ min(\_time)\ as\ earliest\_hit,\\ &\quad\ max(\_time)\ as\ latest\_hit\ by\ uri,\ status\end{aligned}$$

The search is run every hour on the previous hour's data. To clarify the first segment of the above search, an event type is basically a predefined search we can run to tag the events. In this case, the web-traffic-external eventtype tag is defined to capture all events from our Apache logs and limit results to web traffic from non-Splunk sources (i.e. filter out internal clientips). The daily summary index is populated based on the hourly summary index with the following search run every day over the previous day's data:

$$\begin{aligned}&index = summary\_hourly \mid stats\ sum(hits)\ as\ hits,\\ &\quad min(earliest\_hit)\ as\ earliest\_hit,\\ &\quad max(latest\_hit)\ as\ latest\_hit\ by\ uri, status\end{aligned}$$

We search against summary_hourly or summary_daily to calculate the number of hits per uri by status category as follows for any given time period:

$$\begin{aligned}&index = summary\_hourly \mid eval\ status =\\ &\qquad toString(floor(status/100)) + \text{“}xx\text{”}\\ &\quad \mid stats\ sum(hits)\ as\ \text{“}totalcount\text{”}\ by\ uri,\ status\end{aligned}$$

Table 4 shows the performance of running the above search on 5 months of www.splunk.com web logs, from

summary granularity	combined good and bad status time (hr:min:sec)	good status time (hr:min:sec)	bad status time (hr:min:sec)
hourly summary	04:05:35	03:41:00	00:16:56
daily summary	01:21:06	01:16:36	00:04:57

Table 4: Comparison of search performance for calculating number of hits by uri and status using a single summary for good and bad status codes vs using separate summary indexes for good and bad status codes.

January 1, 2011 to May 31, 2001. There is almost 4x performance improvement from using daily summaries over hourly summaries. It is common for web analytics reports to examine successful traffic separately from traffic that resulted in bad status codes. Thus, to make the searches more realistic and to understand the impact of table cardinality on search performance, we performed an additional experiment to create separate summaries for traffic with good status codes (i.e. 200s) and traffic with bad status codes (i.e. 300s to 500s). The search to populate the hourly summary for good traffic included a constraint on the raw data specifying $status >= 200$ and $status < 300$. The search to populate the hourly summary for the bad traffic included a constraint specifying $status >= 300$. We created corresponding separate daily summaries for good and bad traffic.

Based on performance results in Table 4, it is beneficial to separate out bad traffic into its own summary index. Searching for bad traffic against the combined good and bad traffic single summary table would require post processing to select the bad status codes, adding time in computing statistics for the good status codes as well. The performance gain isn't as significant for good status codes, primarily because the number of good status events is orders of magnitude more than the number of bad status events. Thus, it is important to try to minimize the cardinality of data summaries by partitioning non-overlapping search data when possible.

5 Related Work

There is much prior art in data summarization that we embrace and extend in Splunk. The database community has volumes of research on creating materialized views, which are essentially summary tables for the raw data [10]. The concept has even extended to modern MapReduce frameworks, especially Hive, which is noted for its SQL-like interface for MapReduce that enables data summarization among other features [1]. While both traditional RDBMS and Hive expose summarization, they both suffer the limitation that the data ingested must be structured and have a pre-specified schema to form the tables. As a result, they are not convenient for data sources that change over time, as is the case with web logs. Recent work on summarization for log data focused on reducing the data by grouping events into classes; this granularity of summarization is not conducive to obtaining trends and statistics for the web log data.

Much research has gone into analyzing web logs and detecting anomalous behavior [9, 11]. While a majority of this work relies on statistical machine learning techniques, a certain degree of log preprocessing or parsing is inevitable before the algorithm can consume the data. As described in [8], Splunk simplifies data munging and preprocessing, enabling anomaly detection via comparing summarized metrics over time rather than mining voluminous raw data of high-traffic production websites.

6 Conclusion

In this paper, we presented an approach to web analytics that relies on data summarization. We demonstrated the expressiveness of the Splunk search language for creating summary indexes and efficiently reporting on web log data. We strongly believe that Splunk immensely facilitates web-related business and operational insights.

While we demonstrated feasibility and performance of analytics with multi-level summarization, we have yet to assess what user interfaces are required to simplify summarization. It appears desirable to perform summarization with limited user interference, perhaps only to specify summarization granularity. However, we must systematically understand such trade-offs based on the volume of web log data per and the resource requirements for generating the summaries for each environment.

The uses for Splunk are much broader than web analytics. Web logs are just one example of business-critical time series data. There are many other equally valuable datasets that are commonly mined for insights. Some examples include call detail records and CRM data. Splunk already allows users to seamlessly index and search these semi-structured and unstructured time series data and temporally overlay heterogeneous datasets.

One significant area of future work includes exploring data summaries as direct input for machine learning algorithms. Currently, the summaries we generate are convenient for human readable reports. As Splunk moves more towards a data preprocessor for machine learning algorithms, it is equally important to evaluate alternate summary data layout and retrieval mechanisms for machine consumers. We believe this is a promising area of future work that would benefit data mining and analytics.

References

[1] Apache Hive Project. http://wiki.apache.org/hadoop/Hive.

[2] Apache HTTP Server Project. http://httpd.apache.org.

[3] Google Analytics. http://www.google.com/analytics.

[4] Google Urchin. http://www.google.com/urchin.

[5] IIS. http://www.iis.net.

[6] NGiNX. http://wiki.nginx.org.

[7] Omniture Site Catalyst. http://www.omniture.com/en/products/online_analytics/sitecatalyst.

[8] BITINCKA, L., GANAPATHI, A., SORKIN, S., AND ZHANG, S. Optimizing data analysis with a semi-structured time series database. In *Proceedings of the 2010 workshop on Managing systems via log analysis and machine learning techniques* (Vancouver, Canada, 2010), SLAML'10, USENIX Association, pp. 7–7.

[9] BODIK, P., FRIEDMAN, G., BIEWALD, L., LEVINE, H., C, G., PATEL, K., TOLLE, G., HUI, J., FOX, O., JORDAN, M. I., AND PATTERSON, D. Combining visualization and statistical analysis to improve operator confidence and efficiency for failure detection and localization. In *In Proceedings of the 2nd IEEE International Conference on Autonomic Computing (ICAC 05* (2005), IEEE Computer Society, pp. 89–100.

[10] HELLERSTEIN, J. M., AND STONEBRAKER, M. *Readings in Database Systems: Fourth Edition*. The MIT Press, 2005.

[11] LI, W., AND GORTON, I. Analyzing web logs to detect user-visible failures. In *Proceedings of the 2010 workshop on Managing systems via log analysis and machine learning techniques* (Vancouver, Canada, 2010), SLAML'10, USENIX Association, pp. 6–6.