

Optimizing Data Analysis with a Semi-structured Time Series Database

Ledion Bitincka, Archana Ganapathi, Stephen Sorkin and Steve Zhang
Splunk Inc.

Abstract

Most modern systems generate abundant and diverse log data. With dwindling storage costs, there are fewer reasons to summarize or discard data. However, the lack of tools to efficiently store and cross-correlate heterogeneous datasets makes it tedious to mine the data for analytic insights. In this paper, we present Splunk, a semi-structured time series database that can be used to index, search and analyze massive heterogeneous datasets. We share observations, lessons and case studies from real world datasets, and demonstrate Splunk's power and flexibility for enabling insightful data mining searches.

1 Introduction

There is tremendous growth in the amount of data generated in the world. With decreasing storage costs and seemingly infinite capacity due to cloud services such as Amazon S3 [3], there are fewer reasons to discard old data, and many reasons to keep it. As a result, challenges have shifted towards extracting useful information from massive quantities of data.

Mining a massive dataset is non-trivial but a more challenging task is to cross-correlate and mine multiple datasets from various sources. For example, a datacenter monitors data from thousands of components; the log format and collection granularities vary by component type and generation. The only underlying assumption we can make is that each component has a notion of time, either via timestamps or event sequences, that is captured in the logs. As the quantity and diversity of data grow, there is an increasing need for performing full text searches to mine the data.

Another challenge is that a large fraction of the world's data is unstructured, making it difficult to index and query using traditional databases. Even if a dataset is structured, the specifics of the structure may evolve with time, for example, as a consequence of system upgrades

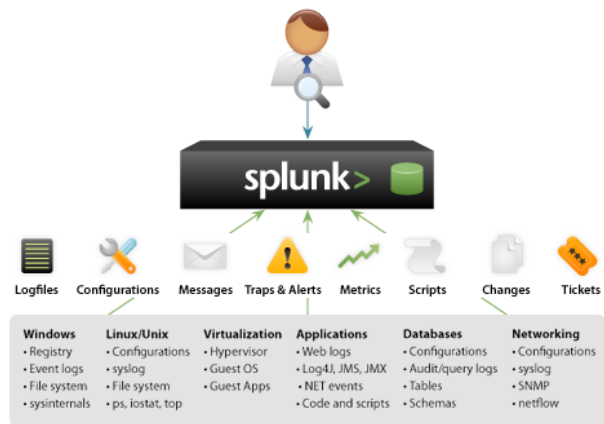


Figure 1: Overview of the Splunk platform

or more/less restrictive data collection/retention policies.

In this paper, we make a case for using a semi-structured time series database to analyze massive datasets. We describe Splunk, a platform for indexing and searching large quantities of data. Splunk accepts logs in any format and allows full text searches across various data sources with no preconceived notions of schemas and relations. Figure 1 shows an example of the wide variety of data Splunk can accept from a heterogeneous set of components.

We share observations from real world deployments with heterogeneous datasets in Section 2 and evaluate existing solutions to store and analyze these datasets. In Section 3 we provide a detailed overview of Splunk, specifically describing formatting and storage considerations for heterogeneous data. We also describe characteristics of a search language that simplifies cross-correlating and mining these datasets. In Section 4, we show examples from real case studies that use simple Splunk searches for gaining insights from data and also discuss common machine learning tasks that are easy to express as Splunk searches. Section 5 concludes.

2 Background and Related Work

In this section, we describe key observations that led to our decision to use a semi-structured time series database. We then describe existing solutions that attempt to address these observations.

2.1 Observations from Heterogeneous Data Sources

We made three key observations from our experiences with a wide variety of data sources and customer deployments, described below.

Observation 1: *Massive datasets are almost always timestamped, heterogeneous, and difficult to fit into traditional SQL database.*

Traditionally, datasets represent state information. They snapshot properties of real-world or virtual objects at some point in time. Some well-known examples of such stateful datasets include store inventories, airline reservations, and corporate personnel data. While these datasets often grow as large as billions of entries, they are bounded in size by real-world limits on the objects they represent. For example, a personnel database cannot grow larger than 7 billion entries as there are fewer than 7 billion people in the world. In addition, there is usually a set of properties (e.g. name, address, salary) for each entry in the database. While the values for each property vary from entry to entry and can evolve over time, the set of properties themselves is effectively fixed at the time the database is designed. Therefore, we refer to these as structured datasets. The general techniques for analyzing structured data are well established and the database community has provided a wealth of specialized tools for this purpose [8]. SQL-query based relational databases have served these structured datasets well.

In contrast, most machine generated logs, such as syslog and web-server logs, are unstructured text files. While the text may have some loosely implied structure, the specific structure varies across systems and is always subject to frequent and unexpected changes. This type of data typically represents a complete history of events over time rather than a snapshot in time. Consequently, machine generated logs are commonly several orders of magnitude larger than structured datasets. In addition, each entry is usually characterized by a timestamp of when the associated event occurred. As an example, popular web services such as Google and Facebook generate billions of web-server log records each day. Each record captures a single HTTP request and contains a timestamp of when each request was received and/or processed. Due to the large scale and temporal organization of log entries, traditional analysis techniques are

often unsuitable for these datasets. Thus despite being goldmines of information, these logs are rarely explored and often deleted to free up storage space.

Observation 2: *Time is the best correlator for heterogeneous data sources.*

Large computing environments, ranging from corporations to governmental and academic institutions, generate tens to hundreds of different types of log data collected from thousands of components (data sources). Due to the unstructured nature of this data, there is usually no unique identifier (primary key) that can be used to join entries across different logs. For example, application server logs often contain Java stacktraces that are directly or indirectly triggered by requests from a web-server. However, these logs contain no explicit identifiers to associate the stacktrace with a specific web-server request. Each event is usually timestamped and thus the time at which each event occurs is often the only piece of information we can use to correlate various types of unstructured log data. Traditional relational database semantics are ineffective for temporal correlation because related events merely have timestamps that are “close” to one another and rarely have the exact same timestamp across various logs.

Observation 3: *Time is of the essence.*

In addition to being the best correlator of unstructured timestamped data sources, time is also essential for data management and search optimization. Analysis is often constrained to data from a particular time range rather than exhaustively across all data. In addition, data from more recent time ranges are typically prioritized over older data. Therefore, it is important to optimize for computation on more recent data. Although declining storage costs make it economical to keep many terabytes and even petabytes of data, fast storage technology, such as SSD or ultra-fast disks, are relatively more expensive. Therefore, a time-sensitive data store should preferentially store more recent data on faster storage, when available, and push older data onto cheaper storage devices. Traditional databases can easily store timestamps, but the value of the timestamp field is not considered when storing the entry and thus it is difficult to optimize for more recent data. Furthermore, analysis over any bounded time range is more efficient if the underlying data is segregated by time.

The above observations, in addition to many other compelling usage scenarios, advocate the need for a semi-structured time series database such as Splunk.

2.2 Related Work

2.2.1 Relational database management solutions (RDBMS)

Converting from the semi-structured format of machine generated data to the dense and homogeneous structure demanded by traditional relational database tables is typically achieved by an Extract, Transform and Load (ETL) procedure. This procedure is often problematic for ad hoc analysis tasks in machine generated data sets. The designer of the ETL procedure must anticipate the gamut of questions to be asked of the data and correctly extract the rows and columns of the table. Such pre-planning is infeasible for two reasons. First, in many systems, there is no complete catalog of all the messages that may be recorded. That is, never-seen-before messages may present themselves exactly when new problems occur. Second, the number of distinct messages in a large system can be prohibitively large and correctly extracting them all up front is often impractical.

By using a flexible retrieval-time schema, these two problems are largely avoided. A new parsing rule can be added as and when a new message is discovered. Moreover, the system's operator has more context when determining how to parse the message.

Another common problem encountered when using a traditional RDBMS for storing machine generated time series data is defining a retention policy. This is important both from the storage footprint perspective and for legal and compliance reasons. Oracle provides a technique called partitioning to segment data into mutually exclusive and jointly exhaustive segments that behave as a monolithic structure [7]. Using partitioning in Oracle allows the operator to specify time ranges to segment the data into. However, this approach requires apriori understanding of how data will arrive.

2.2.2 Map Reduce implementations for data mining

The Apache Hive project [1] also seeks to put a general purpose, scalable interface on a semi-structured data warehouse. Hive provides ETL (Extract, Transform and Load), schematization and analysis of massive amounts of data stored in a Hadoop distributed filesystem (HDFS). Similar to Splunk, Hive uses the MapReduce paradigm, with Hadoop as the job management engine. Hive is designed primarily as a batch processing system and thus queries are not expected to be real-time for the operator. According to the Hive project, even the smallest jobs can take on the order of several minutes. Such lengthy job execution times are unattractive for IT and operations troubleshooting use cases where the time taken to solve a specific problem is very important. Additionally, reducing the cycle time for iterating on query

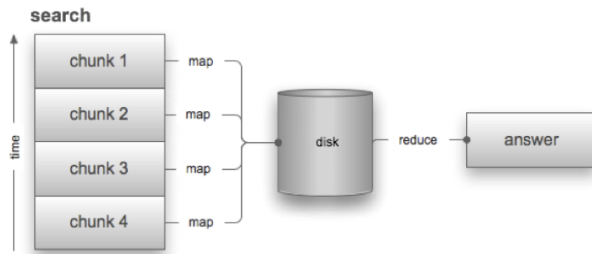


Figure 2: The MapReduce paradigm as used by Splunk

design is very helpful for non-expert users of a system like Splunk.

2.2.3 Processing semi-structured data

Our work is closely related to two research projects.

The LearnPADS tool automatically generates parsers and mining tools specific to a dataset [6]. However, a major drawback to their approach is that the user must provide per-dataset initial descriptions. Requiring a priori knowledge of data formats prevents PADS from scaling to hundreds, let alone thousands, of data sources.

The IBM autonomic computing toolkit's correlation engine allows users to correlate data from two different sources [10]. However, two major limitations of this work are that it only accommodates IBM proprietary log formats, such as Websphere and DB2 logs, and does not scale to correlate more than two data sources at a time.

The Splunk platform can provide the functionality of both these tools and simultaneously scale to thousands of data sources, requiring no prior knowledge/assumptions about data formats. As a result, we can easily build applications on top of Splunk to monitor and provide insights.

3 Semi-structured Time Series Database

Splunk provides users the ability to aggregate and analyze large quantities of data by implementing the MapReduce *divide and conquer* mechanism on top of an indexed datastore [13]. Figure 2 summarizes how Splunk searches are partitioned into *map* and *reduce* phases. Each node (search peer) first partitions the data into mutually exclusive time spans (as opposed to application specific or arrival-based chunks in traditional MapReduce implementations [5]). The map function is applied per time span. The statistics output from these map functions are written to disk. Upon analyzing the entire data set, the reduce function is run on each set of statistics to produce the search result. To generate previews for real-time analysis, we can periodically run the reduce stage without waiting to receive statistics from all mappers.

Whereas traditional MapReduce implementations [5] require the end user to write custom code to leverage their dataset (expressing processing in terms of map and reduce functions or writing queries in SQL like format), Splunk provides an easy-to-use search language, further facilitating realtime data analytics.

Within Splunk, time based partitioning is a first class concept. Without specialized configuration, data loaded into Splunk is automatically partitioned by time so that searches over small time ranges are faster (I/O operations are only performed against the partitions that intersect the query target time) and data can be aged out to secondary or archival storage.

3.1 Optimizing Data Format

In this section, we describe elements of the Splunk data format that have been optimized to address the observations in Section 2.1.

3.1.1 De-normalize to enable better MapReduce

To work around expensive disk space, data normalization effectively minimizes disk usage by eliminating repetitive information. In traditional relational databases, normalization involves removing repeated fields from a table and creating a separate table for that data. However, minimizing data duplication while improving update performance comes at a high data retrieval cost, primarily because related data is stored in different locations. The decrease in storage costs has prompted leading database vendors to promote data structure de-normalization as a way to avoid costly table joins and dramatically improve data retrieval speed. De-normalization typically leverages data redundancy and/or spatial locality to achieve improved read performance.

The massive amount of available data coupled with tremendous growth in data generation rates will unquestionably require distributed data warehousing and processing. MapReduce has proven to be an easy to understand and highly scalable approach to massive distributed data processing. A major drawback of a normalized schema is that it makes distributed processing via MapReduce nearly impossible or ineffective. For instance, with normalized records, each map node in a MapReduce implementation must have access to exclusive work units that no other map node uses. Alternately, if these work units are constructed by accessing multiple data stores shared among all map nodes, a bottleneck will be introduced into the system, making the entire implementation dependent on the performance of the shared system. On the other hand de-normalized records do not impose such limitations and lend themselves naturally to MapReduce distributed data processing.

3.1.2 Timestamp and break data into events

To provide maximum flexibility we have chosen to minimize the amount of data processing we perform at indexing time. The two most important types of data processing during indexing time are (i) event boundary detection and (ii) event timestamping.

There are numerous tools that process single line data including, most notably, text processing commands in UNIX. However, many applications' log events span multiple lines. For example, Java stack traces, Windows application/system/security events, and some mail servers generate multi-line log entries. Event boundary detection enables us to create minimal working units, which are then stored in a time based proprietary database. Splunk provides automatic event boundary detection that uses timestamps to break the text stream into separate events. Users can also define custom event boundaries through configuration changes.

As time is usually the best and often only variable that can be used to correlate events, it is extremely important for Splunk to assign the correct timestamp to an event. Splunk provides both automatic timestamp detection and an extensive set of configurations for specifying a custom format to assign event timestamps.

3.1.3 Structure and keyword indexes

A major advantage of Splunk is its ability to process highly unstructured data while allowing users to efficiently search and report on this data. However, even the most unstructured logs have some very basic structure which can be used to apply a retrieval time schema to the data. All events in Splunk are guaranteed to have some basic structure consisting of at least the following fields: *time*, *source*, *sourcetype*, *host* and *event text*. *Sourcetype* is a concept that is used in Splunk to group similar data sources, and is determined by looking at rules on the source or via a clustering algorithm on data content. Based on the value of some of these fields users can choose to rely on automatic field extraction or provide custom parsing rules including, but not limited to, regular expression-based rules and delimiter-based rules.

The Splunk time series database is optimized to perform efficient retrieval based on time and keywords. Although Splunk applies a schema at search time, the schema is made accessible to users during the search. To achieve this access we split our search into two phases - retrieval and filtering. Before the search starts executing, we determine the time range required for the search and a set of keywords that all matching events must have. The retrieval phase uses these parameters to query the database for a superset of the required results. Upon completion of the retrieval phase, we have enough information to apply the schema and enter the filtering phase,

```

All Events:
06-07-2010 12:49:21.955 INFO databasePartitionPolicy - No databases found starting fresh !
06-08-2010 11:14:00.836 WARN pipeline - Exiting pipeline parsing gracefully: got eExit from processor sendOut
06-10-2010 06:29:53.749 INFO Metrics - group=queue, name=typingqueue, max_size=1000, filled_count=0, empty_count=46, current_size=0, largest_size=2, smallest_size=0
06-10-2010 06:30:24.449 INFO Metrics - group=pipeline, name=indexerpipe, processor=indexandforward, cpu_seconds=0.000000, executes=119, cumulative_hits=389000

Retrieval Results:
06-08-2010 11:14:00.836 WARN pipeline - Exiting pipeline parsing gracefully: got eExit from processor sendOut
06-10-2010 06:30:24.449 INFO Metrics - group=pipeline, name=indexerpipe, processor=indexandforward, cpu_seconds=0.000000, executes=119, cumulative_hits=389000

Filtering Results:
06-10-2010 06:30:24.449 INFO Metrics - group=pipeline, name=indexerpipe, processor=indexandforward, cpu_seconds=0.000000, executes=119, cumulative_hits=389000

```

Figure 3: Example of keyword indexing.

which returns the required results.

An example best demonstrates how Splunk search uses the keyword index. Assume that our database consists of the four events seen in Figure 3. Note that these events have been indexed using their timestamp and text tokens in the database. A user then issues the following search to be run for the month of June 2010:

```
search group = "pipeline" | ...
```

Note that the field *group* is being used as a schema specific constraint.

During the retrieval phase, we query our database using the following parameters:

```
Time range: 06-01-2010 00:00:00 to 07-01-2010 00:00:00
```

```
Keywords: pipeline
```

Since all four log entries satisfy the temporal constraints of the search, the retrieval phase returns two of the four events that match the *pipeline* keyword criterion. Subsequently, the filtering phase performs field extraction (in this example, automatic field extraction, as described in Section 3.3, suffices), applies schema related work such as aliasing, event typing and lookups, and filters the events using schema specific constraints. Our final result for this example consists of a single event.

3.2 Splunk Search Language

A crucial component of Splunk for data analytics is the ability to search across the various data sources. Splunk has developed a search language which is aimed at being powerful yet simple. The search language is based on the UNIX concept of pipes and commands. Just like in the UNIX environment, where users are allowed to extend the basic set of available commands in a shell, users of splunk can extend the search language by introducing their own custom search commands.

Figure 4 shows an example of a search to find all events that contain the term “error” and then compute a table with two columns - host, count.

The search commands can be logically grouped into two major groups:

- streaming - operate on single events
- aggregating - operate on the entire event set

```
search "error" | stats count BY host
```

Sample search results:

host	count
example.com	123
localhost	231

Figure 4: Search example finds all events that contain the term “error” and computes a table with host and count columns.

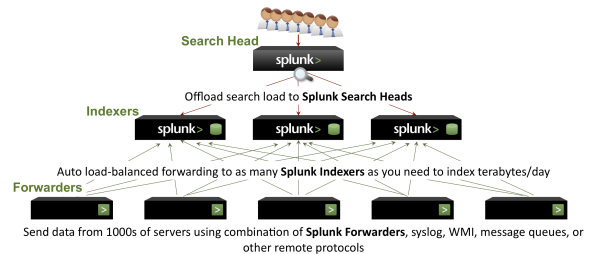


Figure 5: Splunk search in distributed environments.

Before explaining how the streaming and aggregating commands fit in the MapReduce like implementation we briefly describe the components of a splunk distributed deployment. Note that a simple Splunk deployment has all functions in a single binary on a single machine. The distributed environments generally comprise of three components:

- forwarders - responsible for gathering event information and forwarding it to the indexers
- indexers - responsible for indexing received events and servicing queries from search head(s)
- search head(s) - responsible for providing a single query point(s) for the entire deployment

Figure 5 shows a schematic of how these three components interact with one another. During a search only the search head and indexers are involved. The search head is responsible for analyzing the given search to determine what part can be delegated for execution by in-

dexers (also known as search peers) and what part needs to be executed by the search head. Streaming commands can be trivially delegated to the indexers. Conversely, aggregating commands are more complex to distribute. To achieve better computation distribution and minimize the amount of data transferred between search peers and the search head, many aggregating commands implement a map operation which the search head can delegate to the search peers while executing the reduce operation locally.

Figure 6 shows how the original search in Figure 4 is split into two parts: one to be executed by peers and one to be executed by the search head.

```
Original search:
search "error" | stats count BY host

Sent to peers:
search "error" | prestats count BY host

Executed by search head:
Merge the prestats results received from peers (reduce)
```

Figure 6: Example of a search being split into two parts.

In the above example the search peers are responsible for counting the results by host and sending their results to the search head for merging - achieving both computation distribution and minimal data transfer.

3.3 Logging Best Practices

Logging formats vary considerably among different applications as well as between different components of a single application. Syslog [11] has brought some standardization to logging, but it falls short in recommendations on how to effectively format the log message body. Over the last decade there have been many proposals for standardizing the format of the log message body, however for a number of reasons none of them have gained industry wide support yet. One of the most compelling reasons hindering industry from adopting a logging standard is the high cost associated with changing the format. A log format change will invariably break a majority of the tools that are in place for parsing and processing those logs. There have been several proposals for log message standardization [12, 4, 9, 2].

Although Splunk can perform keyword searches on any data format that it has indexed, it processes data formatted in a certain way better than others. As we describe in Section 3.1.3, Splunk is able to apply automatic and user configured schema to events at retrieval time. In this section, we describe the logging format for which the automatic schema is optimized as well as present some tips on how to format log messages.

Splunk automatically parses events that contain fields of the form: `< fieldname >= " < fieldvalue > "` The quotes can be dropped if the field value does not contain any breaker characters, that is one of `"\n\t,&;|"`.

The last line of the example in Figure 3 shows one such log message emitted by Splunk's metrics collection subsystem. Using a logging format as described above relieves the user from the task of maintaining the schema they apply to their events. This logging format is human readable yet flexible enough for applications to add/remove fields from log messages without triggering schema changes.

Although machine generated data is primarily intended for use by machines it is important to use a logging format that is human readable where each event contains enough information to allow an operator to make sense of the message. Here are some common tips to improve logging readability and simplify parsing.

1. Include field names in the message. For example:

```
"06-09-2010 03:34:54 PST INFO src_ip=10.5.36.5
src_port=2256 dst_id=10.25.36.38 dst_port=5586
bytes=1236584"
```

is much easier to read than

```
"06-09-2010 03:34:54 INFO 10.5.36.5:2256 10.25.36.38:5586
1236584"
```

2. Ensure each event contains an unambiguous time stamp, preferably also containing timezone information.
3. Include any unique ids that the event relates to, such as transaction id, user id, product id, message id.
4. To cross-correlate multiple data sources standardize on variable names across different components. For example, standardize on using either of: `s_ip`, `source_ip`, `src_ip` across all the logging applications to denote a source IP.

4 Real World Search Examples

In this section, we present two case studies from real customers who solved critical IT operational problems using a Splunk search, and also discuss more general data mining searches that Splunk facilitates.

4.1 Case Studies

Modern IT organizations are faced with the non-trivial task of managing thousands of diverse components interacting in complex ways. These operational challenges are rooted in three ground truths:

1. **change is unpredictable.** IT operations are severely affected by unanticipated load spikes and must quickly detect and handle them to avoid downtime. Periodic changes to workload volume and distribution may also be attributed to diurnal patterns of daily, weekly or seasonal activity.
2. **upgrades are inevitable.** Hardware and software components are periodically upgraded and/or re-configured to accommodate new features and grow the system. Furthermore, new metrics may need to be extracted from existing components to accommodate protocol upgrades, from IPv4 to IPv6 for example.
3. **failures are a fact of life.** Hardware ages, software is buggy, and humans make mistakes. Therefore, failures cannot be prevented but at best be detected quickly, masked or mitigated.

In both case studies below, Splunk helped the customers aggregate and mine data to cross correlate and search for insights across various data sources. We demonstrate simplicity and powerfulness of these customers' Splunk searches. We abstract away the original names of these customers due to privacy considerations.

4.1.1 IT powerhouse for a major industry

This organization provides the infrastructure for all the transaction processing needs of a major industry. They maintain records of every element of every end user transaction to each of their customer websites, collect information per request received and response sent. They also log telemetry about when, where and how each transaction was processed. They use Splunk to index 35 to 50 GB of data per day. Data is aggregated from about 50 components and the data formats include, but are not limited to, raw XML logs, $\langle key, value \rangle$ pairs, ASCII logs, system logs and database transaction logs from Oracle and MySQL.

Since each transaction passes through multiple sub-systems, a major challenge faced by this organization is to identify the statistical and transactional data it leaves behind on each component in its path. The organization can use Splunk to obtain a comprehensive map of each transaction's imprint on the infrastructure and consequent system load and health metrics as follows:

Example customer search: "List my top 100 customers in terms of CPU time consumed:"

```
search *
| stats sum(cpu_seconds) AS totalcpu BY customer
| sort 100 - totalcpu
```

Due to the ease of searching historical data per customer or end-user, the organization can also develop more accurate predictive models to assist with business intelligence and provisioning.

4.1.2 IT organization at a University

This university IT organization provides infrastructure and support for tens of thousands of students and thousands of faculty campus-wide. They manage about 600 devices and index 40 GB of data daily using Splunk. Their data sources include firewall events, kerberos and LDAP authentication logs and antivirus logs, to name a few.

A compelling use case of Splunk for this organization is the ability to search for all events associated with a single IP address. In the event that a machine acquires a virus and is generating malicious traffic, the IT organization can use Splunk to cross-correlate server events and network traffic with anti-virus logs and identify which IP address is the culprit.

Example customer search: "Get the login action server events for the IP addresses mentioned in the last 100 anti-virus alert messages:"

```
search sourcetype = server
  action = login [search sourcetype = antivirus
                 ALERT | head 100 | fields ip]
```

This example utilizes a subsearch, denoted within square brackets, to first obtain the last 100 antivirus alert messages. Subsequently, the subsearch results are searched for login action server events.

Using nested subsearches, this organization can also look up the owner who registered the infected node and convey corrective actions or reprimand malicious intent.

4.2 Data Mining using Splunk

We demonstrated how simple it is to cast the above two organizations' pain point problems as Splunk searches. Additionally, we believe Splunk is valuable for general data mining beyond the realm of IT. Splunk provides simple mathematical and statistical operations such as averages, variance, and order statistics. Many business intelligence mining use cases can be satisfied using simple combinations of these primitives.

Perhaps more importantly, Splunk provides a platform for easy, efficient data cleaning and munging. Even when researchers are exploring novel machine learning techniques, one of the most difficult and time consuming tasks is preparing the data. For example, most clustering algorithms require a dense table of continuous metrics. Categorical data must be recast to be continuous

and rows with missing data point usually need to be discarded or interpolated. On the other hand, learning algorithms like decision trees and naive-Bayes often require their input be categorical instead of continuous. Regardless of the technique, Splunk provides a simple way of preparing massive data sets for analysis. Furthermore, Splunk can normalize variable names at search time, requiring no manual preprocessing before cross-correlating datasets from multiple sources.

Below, we demonstrate a few examples of data mining and preparation tasks that are easily expressed using Splunk.

Data cleaning: remove data points that do not have a score field.

```
search score = *
```

Data munging: combine two different sources that have a different format for the score field. Source X scores are based on a 0-100 scale, and source Y uses letter grades and needs to be changed to a 0-100 scale.

```
search source = X OR source = Y
  | eval score = if(source = "X", score,
    case(score = "A", 100, score = "B", 85,
      score = "C", 70, score = "D", 60,
      score = "F", 0))
```

Outlier detection: find scores more than 3 standard deviations more or less than the average.

```
search score = *
  | eventstats avg(score) AS avg_stdev(score) AS stdev
  | where (score > avg + 3 * stdev)
  OR (score < avg - 3 * stdev)
```

Correlation: covariance of score and income. Note that Splunk interprets the first line of the search below to use an implicit AND to obtain all scores and all incomes.

```
search score = * income = *
  | stats avg(eval(score * income)) AS avg_prod
    avg(score) AS avg_score
    avg(income) AS avg_income
  | eval cov = avg_prod - avg_score * avg_income
```

Clustering: simple kmeans of score, income, age. Again, Splunk inserts an implicit AND to search all scores, incomes and ages.

```
search score = * income = * age = *
  | kmeans k = 10 score income age
```

5 Conclusions

In this paper, we presented Splunk, an analytics platform for massive datasets. We discussed why and how a semi-structured time series database is the most appropriate mechanism for cross-correlating and mining heterogeneous datasets. We demonstrated the power and ease of using the Splunk search language for solving IT operational problems.

Domain knowledge is crucial for successful data analysis. However, domain experts often lack familiarity with statistical/machine learning techniques to leverage them well. Splunk can abstract away the statistical expertise and thus enables domain expertise, rather than statistical/machine learning proficiency, to drive the analysis of any particular dataset.

Many researchers have successfully leveraged statistical machine learning techniques to gather system insights from data. Such techniques require a dominant fraction of our time to be spent on collecting, parsing, and cleansing the data, and only a small fraction is attributed to applying the algorithm itself. Furthermore, data mining research is often targeted at a specific dataset, and may not generalize to other types of data, let alone other systems. Thus, we feel it is important to target future work to improve the infrastructure for performing such data driven research rather than focus on algorithmic improvements. This new direction would have a broader impact on the research community and enable easier and more insightful data mining.

Splunk can be the foundation for newer infrastructures, but also has the potential to serve as the overarching infrastructure to develop new data mining techniques.

References

- [1] Apache Hive Project. <http://wiki.apache.org/hadoop/Hive>.
- [2] WELF. <http://www.m86security.com/kb/article.aspx?id=10899cNode=5R6Q0N>.
- [3] AMAZON.COM. Amazon Simple Storage Service (Amazon S3). <http://aws.amazon.com/s3/>.
- [4] ARCSIGHT. CEF. <http://www.arcsight.com/collateral/CEFstandards.pdf>.
- [5] DEAN, J., AND GHEMAWAT, S. MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM* 51, 1 (2008), 107–113.
- [6] FISHER, K., WALKER, D., AND ZHU, K. Q. LearnPADS: automatic tool generation from ad hoc data. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (New York, NY, USA, 2008), ACM, pp. 1299–1302.
- [7] FLINT, A., AND COOKSON, I. Data Warehousing on Oracle RAC Best Practices. White paper, Oracle Corporation, October 2008.
- [8] HELLERSTEIN, J. M., AND STONEBRAKER, M. *Readings in Database Systems: Fourth Edition*. The MIT Press, 2005.

- [9] IBM. CBE. <http://www-128.ibm.com/developerworks/webservices/library/ac-cbe1/>.
- [10] JACOB, B., LANYON-HOGG, R., NADGIR, D. K., AND YASSIN, A. F. A Practical Guide to the IBM Autonomic Computing Toolkit. IBM Redbooks, April 2004.
- [11] LONVICK, C. The BSD Syslog Protocol, 2001.
- [12] MITRE, SPLUNK AND LOGLOGIC. CEE. <http://cee.mitre.org/docs/Common.Event.Expression.White.Paper.June.2008.pdf>.
- [13] SORKIN, S. Large-Scale, Unstructured Data Retrieval and Analysis Using Splunk. Technical paper, Splunk Inc., 2009.