splunk>

Splunk Technical Paper:

Large-Scale, Unstructured Data Retrieval and Analysis Using Splunk.

An Easier, More Productive Way to Leverage the Proven MapReduce Paradigm

Stephen Sorkin, VP of Engineering

Splunk Inc. | 250 Brannan Street | San Francisco, CA 94107 | www.splunk.com | info@splunk.com

Background:

Splunk is a general-purpose search, analysis and reporting engine for time-series text data, typically machine data. Splunk software is deployed to address one or more core IT functions: application management, security, compliance, IT operations management and providing analytics for the business.

Splunk versions 1-2 were optimized to deliver a typical Web search experience, where a search is run and the first 10 to 100 results are displayed quickly. Any results past a fixed limit were irretrievable, except if some other search criterion were added. Under these conditions Splunk scalability had its limitations. In version 3 Splunk added a statistical capability, enhancing the ability to perform analytics. With this addition, searches could be run against the index to deliver more context.

In response to input from Splunk users and to address the challenges of large-scale data processing, in July 2009 Splunk 4 changed this model by adding the facility to retrieve and analyze massive datasets using a "divide and conquer" approach: the MapReduce mechanism. With version 4, Splunk optimized the execution of its search language using the MapReduce model where parallelism is key.

Of course, parallelizing analytics via MapReduce is not unique to Splunk. However, the Splunk implementation of MapReduce on top of an indexed datastore with its expressive search language provides a simpler, faster way to analyze huge volumes of machine data. Beyond this, the Splunk MapReduce implementation is part of a complete solution for collecting, storing and processing machine data in real time, thereby eliminating the need to write or maintain code—a requisite of more generic MapReduce implementations.

This paper describes the Splunk approach to machine data processing on a large scale, based on the MapReduce model.

What is MapReduce?

MapReduce is the model of distributed data processing introduced by Google in 2004. The fundamental concept of MapReduce is to divide problems into two parts: a map function that processes source data into sufficient statistics and a reduce function that merges all sufficient statistics into a final answer. By definition, any number of concurrent map functions can be run at the same time without intercommunication. Once all the data has had the map function applied to it, the reduce function can be run to combine the results of the map phases.

For large scale batch processing and high speed data retrieval, common in Web search scenarios, MapReduce provides the fastest, most cost-effective and most scalable mechanism for returning results. Today, most of the leading technologies for managing "big data" are developed on MapReduce. With MapReduce there are few scalability limitations, but leveraging it directly does require writing and maintaining a lot of code.

This paper assumes moderate knowledge of the MapReduce framework. For further reference, Jeffrey Dean and Sanjay Ghemawat of Google have prepared an introductory paper on the subject: http://labs.google.com/papers/mapreduce.html. Additionally, the Google and Yahoo papers on Sawzall http://labs.google.com/papers/sawzall.html and Pig http://research.yahoo.com/project/90 provide good introductions to alternative abstraction layers built on top of MapReduce infrastructures for large-scale data processing tasks.

Speed and Scale Using a Unique Data Acquisition/Storage Method

The Splunk engine is optimized for quickly indexing and persisting unstructured data loaded into the system. Specifically, Splunk uses a minimal schema for persisted data – events consist only of the raw event text, implied timestamp, source (typically the filename for file based inputs), sourcetype (an indication of the general "type" of data) and host (where the data originated). Once data enters the Splunk system, it quickly proceeds through processing, is persisted in its raw form and is indexed by the above fields along with all the keywords in the raw event text.

In a properly sized deployment, data is almost always available for search and reporting seconds after it was generated and written to disk; network inputs make this even faster. This is different from Google's MapReduce implementation and Hadoop, which are explicitly agnostic towards loading data into the system.

Indexing is an essential element of the canonical "super-grep" use case for Splunk, but it also makes most retrieval tasks faster. Any more sophisticated processing on these raw events is deferred until search time. This serves four important goals: indexing speed is increased as minimal processing is performed, bringing new data into the system is a relatively low effort exercise as no schema planning is needed, the original data is persisted for easy inspection and the system is resilient to change as data parsing problems do not require reloading or re-indexing the data.

A typical high-performance Splunk deployment involves many servers acting as indexers. As data is generated throughout the network from servers, desktops or network devices, it is forwarded to one of the many indexers. No strong affinity exists between forwarder and indexer, which prevents bottlenecks that arise from interconnections in the system. Moreover, any data can be stored on any indexer, which makes it easy to balance the load on the indexers. Although spreading the data across many indexers can be thought of as a mechanism to increase indexing performance, through MapReduce, the big win comes during search and reporting, as will be seen in the following sections.

Simplified Analysis with the Splunk Search Language

The Splunk Search Language is a concise way to express the processing of sparse or dense tabular data, using the MapReduce mechanism, without having to write code or understand how to split processing between the map and reduce phases. The main inspiration of the Splunk Search Language was Unix shells, which allow "piping" data from one discrete process to another to easily achieve complicated data processing tasks. Within Splunk this is exposed as a rich set of search commands, which can be formed into linear pipelines. The mental model is that of passing a table from command to command. The complication for large-scale tasks is that the table may have an arbitrarily large number of rows at any step of the processing.

The columns in the search-time tables are commonly referred to as fields in this document. Unlike in typical databases, these columns do not exist in the persisted representation of the data in Splunk. Through search, the raw data is enriched with fields. Through configuration, fields are automatically added via techniques like automatic key/value extraction, explicit extraction through regular expressions or delimiters or through SQL

join-like lookups. Fields are also constructed on the fly in search using commands like "eval" for evaluating expressions, "rex" for running regular expression-based extractions and "lookup" for applying a lookup table.

Below are examples of some common and useful searches in the Splunk Search Language:

- `search googlebot`
  This search simply retrieves all events that contain the string "googlebot." This is very similar to running "grep –iw googlebot" on the data set, though this case benefits from the index. This is useful in the context of Web access logs, where a common task is understanding what common search bots are causing hits and which resources they are hitting.

- `search sourcetype=access_combined | timechart count by status`
  This search retrieves all data marked with the sourcetype "access_combined," which is typically associated with web access log data. The data is then divided into uniform time-based groups. For each time group, the count of events are calculated for every distinct HTTP "status" code. This search is commonly used to determine the general health of a web server or service. It is especially helpful for noticing an increase of errors, either 404s which may be associated with a new content push or 500s which could be related to server issues.

- `search sourcetype=access_combined | transaction clientip maxpause=10m | timechart median(eventcount) perc95(eventcount)`
  This search again retrieves all web access log data. The events are then gathered into transactions identified by the client IP address, where events more than 10 minutes apart are considered to be in separate transactions. Finally, the data is summarized by time and, for each discrete time range, the median and 95th percentile session length is calculated.

Formulating a Distributed Task Without Writing a Lot of Code

In alternative MapReduce frameworks like Google's own, or the open source Hadoop, it is the user's job to break any distinct retrieval task into the map and reduce functions. For this reason, both Google and Yahoo have built layers on top of a MapReduce infrastructure to simplify this for end users. Google's data processing system is called Sawzall and Yahoo's is called Pig. A program written in either of these languages can be automatically converted into a MapReduce task and can be run in parallel across a cluster. These layered alternatives are certainly easier than native MapReduce frameworks, but they all require writing an extensive amount of code.

Within Splunk, any search in the Search Language is automatically converted into a parallelizable program (the equivalent of writing code in Hadoop or scripts in Pig or Sawzall). This happens by scanning the search from beginning to end and determining the first search command that cannot be executed in parallel. That search command is then asked for its preferred map function. The fully-parallelizable commands combined with the map function of the first non-parallelizable command are considered the map function for the search. All subsequent commands are considered the reduce function for the search. We can consider two simple examples from the world of Web Analytics to illustrate this concept:

- `search sourcetype=access_combined | timechart count by status`

This search first retrieves all events from the sourcetype "access_combined." Next, the events are discretized temporally and the number of events for each distinct status code is reported for each time period. In this case, "search sourcetype=access_combined" would be run in parallel on all nodes of the cluster that contained suitable data. The "timechart count by status" part would be the reduce step, and would be run on the searching node of the cluster. For efficiency, the timechart command would automatically contribute a summarizing phase to the map step, which would be translated to "search sourcetype=access_combined | pretimechart count by status." This would allow the nodes of the distributed cluster to only send sufficient statistics (as opposed to the whole events that come from the search command). The network communications would be tables that contain tuples of the form (timestamp, count, status).

- `eventtype=pageview | eval ua=mvfilter(eventtype LIKE "ua-browser-%") | timechart dc(clientip) by ua`

Events that represent pageviews are first retrieved. Next, a field called "ua" is calculated from the eventtype fields, which contains many other event classifications. Finally, the events are discretized temporally and the number of distinct client IPs for each browser is reported for each time period. In this case, "search eventtype=pageview | eval ua=mvfilter(eventtype LIKE "ua-browser-%")" would be run in parallel on all nodes of the cluster that contained suitable data. The "timechart dc(clientip) by ua" part would be the reduce step, and would be run on the searching node of the cluster. Like before, the timechart command would automatically contribute a summarizing phase to the map step, which would be translated to "search eventtype=pageview | eval ua=mvfilter(eventtype LIKE "ua-browser-%") | pretimechart dc(clientip) by ua." The network communications would be tables that contain tuples of the form (timestamp, clientip, ua, count).

Temporal MapReduce – How Search Scales on a Single Node

Splunk uses a MapReduce model even when running on a single node. Instead of running the map function on each node in the cluster, Splunk divides the data into mutually exclusive but jointly exhaustive time spans. The map function is run independently on each of these time spans and the outputted sufficient statistics are preserved on disk. Once the entire set of data has been analyzed, the reduce function is run on each of the sets of sufficient statistics.

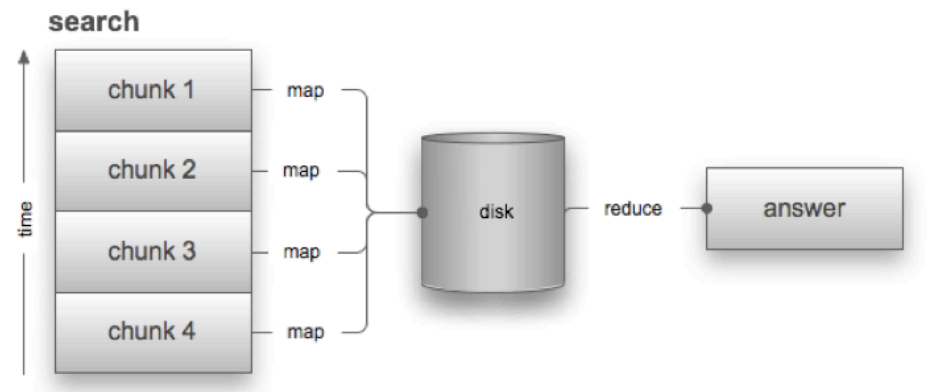

Figure 1:
Temporal MapReduce.

Enhance the Efficiency of the MapReduce Scheme with Previews

To preserve the attractive interactive behavior of search even when running over very large datasets, some enhancements are made over this basic MapReduce scheme when running searches from Splunk. One important change is to run the reduce phase on a regular basis over the sets of sufficient statistics in order to generate previews of the final results without having to wait for the search to complete. This is typically very helpful in refining complicated searches by identifying errors in a search (e.g., misspellings of a field, incorrect values specified, or improper statistical aggregation) before too much time passes and resources are wasted.

Spatial MapReduce – How Search Scales on Additional Nodes

Splunk also provides a facility, like typical MapReduce, to speed computation by distributing processing throughout a cluster of many machines. In Splunk this is called "Distributed Search." After a search has been formulated into the map and reduce functions, network connections are established to each Splunk Indexer in the search cluster. The map function is sent to each of these Splunk instances and each begins processing data using the Temporal MapReduce scheme. As data streams back to the instance that initiated the search, it is persisted to disk for the reduce function.

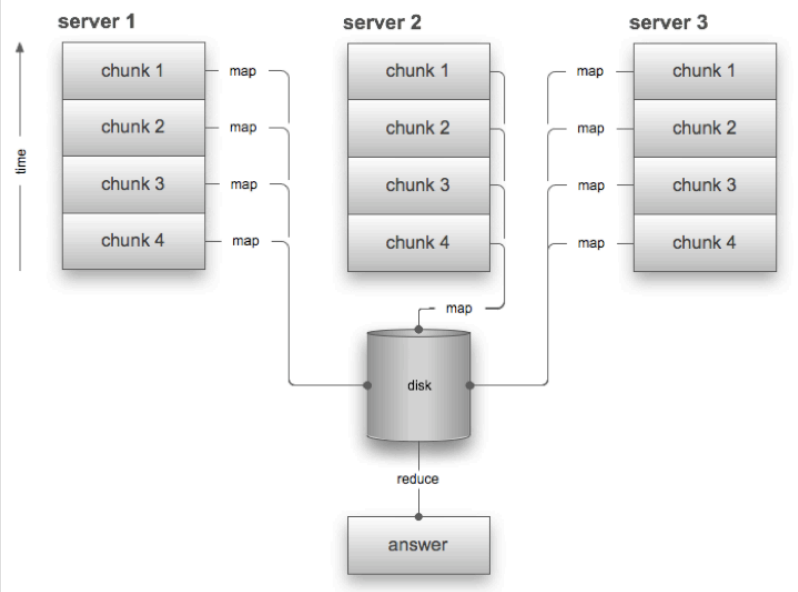

Figure 2: Distributed Search in Splunk.

In general, for pure reporting searches that have a map function that compresses data for network transport, reporting speed is linear with the number of index servers in the cluster. Some searches, however, don't have such a map function. For example, a search for "*" of all the data indexed by every server must be returned to the searching instance for accumulation. Here the instance can become the bottleneck. There are some pure searches that benefit dramatically from the distribution of work. Searches that are dominated by I/O costs (needle-in-a-haystack type searches) and those that are dominated by decompression costs (retrieving many records well distributed in time and not dense in arrival) both benefit dramatically from Distributed Search. In the first case the effective I/O bandwidth of the cluster scales linearly with the number of machines in the cluster. In the second case the CPU resources scale linearly.

For the majority of use cases (i.e., needle-in-a-haystack troubleshooting or reporting and statistical aggregation) scalability is only limited by the amount of indexers dedicated to the processing task.

Achieving Additional Scalability Through Summary Indexing

The problem of analyzing massive amounts of data was addressed in earlier versions of Splunk using a technique called Summary Indexing. The general idea is that relatively small blocks of raw data (datasets between five minutes and an hour in length) are processed on a regular basis and the result of this processing is persisted in a similar time-series index as the original raw data. This can be immediately seen as the map function of MapReduce. Later, when questions are asked of the data, these sufficient statistics are retrieved and reduced into the answer.

This technique is still valuable in Splunk 4 if common questions are repeatedly asked. By running this preprocessing over the data on a regular basis (typically every hour or every day), the sufficient statistics are always ready for common reports.

Comparing Languages: Sawzall and the Splunk Search Language

A key benefit of the Splunk Search Language is the simplicity it provides for common tasks. For illustration, we will consider the first example from the Sawzall paper. This example computes the number of records, sum of the values and sum of the squares of the values.

An example of the Sawzall language follows:

```
count: table sum of int;
total: table sum of float;
sum_of_squares: table sum of float;
x: float = input;
emit count <- 1;
emit total <- x;
emit sum_of_squares <- x * x;
```

Achieving this same task using the Splunk Search Language is far more efficient:

```
source=<filename> | stats count sum(_raw) sumsq(_raw)
```

Note: Published docs from Google tout Sawzall as being a far simpler language than using a MapReduce framework (like Hadoop) directly. According to Google, typical Sawzall programs are 10-20 times shorter than equivalent MapReduce programs in C++.

The efficiency and ease with which Splunk delivers scalability over massive datasets is not confined to this comparison. The contrast becomes sharper if one must properly extract value(s) from each line of the file. Splunk provides field extractions from files by delimiters or regular expressions, either automatically for all files of a certain type or on a search-by-search basis. Additionally, Splunk enables users to learn regular expressions interactively based on examples from within a data set, so the user and product build knowledge over time.

Conclusion – Out-of-the-box Scalability for Big Data with Splunk

When it comes to processing massive datasets the implementation of MapReduce is the performance and scalability model for parallel processing executed over large clusters of commodity hardware. Out of MapReduce, several languages and frameworks have evolved: Google Sawzall, Yahoo! Pig, the open source Hadoop framework and Splunk. While MapReduce is an essential element to scaling the capabilities of search and reporting in Splunk, the out-of-the-box benefits of using Splunk for large-scale data retrieval extend beyond MapReduce processing.

Unlike other MapReduce languages and frameworks that require custom scripts or code for every new task, Splunk utilizes its search language to automate complex processing. The Splunk Search Language makes challenging data analysis tasks easier without requiring the user to control how it scales. With its MapReduce underpinnings, the scalability of Splunk is only limited by the amount of resources dedicated to running Splunk indexers.

Beyond the simplicity of the search language, Splunk provides a universal indexing capability to automate data access and loading. With numerous mechanisms for loading data, none of which require developing or maintaining code, Splunk users are productive quickly.

The speed and efficiency of Splunk are also enhanced through multiple methods of distributing the index (storage) to disk. With the ability to distribute storage across a single node, multiple nodes, or to preview a search in progress, Splunk provides flexible alternatives for optimizing scalability.

Analyzing large datasets is simplified with the Splunk user interface. The Splunk UI, enhanced over several product releases, is highly domain appropriate for the troubleshooting, on-the-fly reporting and dashboard creation associated with large-scale machine data analysis.

Download a Free version of Splunk: http://www.splunk.com/download
For more information on Splunk please visit: http://www.splunk.com

###