Large Scale Splunk Tuning

DAVID LANG

David Lang is a site reliability engineer at Google. He spent more than a decade at Intuit working in the Security Department for the Banking Division. He was introduced to Linux in 1993 and has been making his living with Linux since 1996. He is an Amateur Extra Class Radio Operator and served on the communications staff of the Civil Air Patrol, California Wing, where his duties included managing the statewide digital wireless network. He was awarded the 2012 Chuck Yerkes award for his participation on various open source mailing lists. david@lang.hm

Splunk is a great tool for exploring your log data. It's very powerful, but still very easy to use. At lower log volumes (especially with the 500 MB/day, single-system, no-cost license) it's basically install and use. Because its paid licensing only depends on the amount of new data it sees each day, you can scale its performance and available history for "only" the cost of the systems to run it on. Very few tools can compete with Splunk for unplanned searches over very large volumes of log data. It is very practical to search hundreds of terabytes of log data and get the answer back within a minute or two.

However, as you scale up Splunk to handle these larger log volumes, you move away from the simple "anyone can do anything" install and use model and toward something that's much closer to the traditional database administration model, where you assign a small set of people to become experts in Splunk internals, watching performance, changing the configuration of Splunk, and influencing the queries sent to Splunk.

In the last article in this series [1], I talked about the ways that dashboards and reports can kill your log query system and how to change them to minimize the load they create. In this article, I will be talking about administering and configuring Splunk systems to perform the log searches. I am specifically covering the configuration of the core servers, exploring the conflicting goals that you will need to balance to match your particular data set and workload.

Overview of Splunk Internals

The main tuning configuration settings that you will want to watch are the configuration settings of your Indexes in Splunk. In Splunk "Index" is used in a way analogous to "database" in a traditional DBMS like PostgreSQL, MySQL, or Oracle. You have one instance of the main engine running, and it contains multiple databases/indexes inside it. Data is segregated between the different Indexes (although not as firmly as among different databases in a DBMS because you can issue one query that searches multiple Indexes). Unfortunately, there is no one right way to configure Splunk any more than there is a right way to configure any other DBMS. Everything is a tradeoff, where changing things will improve performance under some conditions and decrease performance in other situations. This article maps out the main configuration changes that you should be looking at, and what the benefits and costs are to provide a set of guidelines to get you started.

The one user visible configuration you will need to manage is defining what logs will end up in what Index. Users can limit their queries to specific Indexes, and user permissions can provide very solid access controls at the Index level. User permissions can provide weaker access controls within an index by automatically appending search terms to any query the user performs. I will go into more detail on the advantages and disadvantages of configuring what logs go to what Index later in the article. The other configuration changes are not user visible.

To start with, I need to go over some low-level details of how Splunk is implemented and related terminology.

Splunk is a MapReduce type of system where one or more search heads provide the search interface for the users and then parse the query, determine which Indexes need to be searched (explicitly specifying them in the search or falling back on the user's default set of Indexes), and dispatch the appropriate sub-queries to the various Indexer machines. The search head then combines the results and does any additional processing that's needed. In most use cases, the bottleneck is going to be in the data retrieval from the Indexer machines far more than the search heads.

When an Indexer receives a search from a search head, it applies timestamp filters. The data in an Index is split up into "buckets" no larger than a per-index configured size. Buckets are identified by the date range contained in that bucket, so Splunk doesn't even look in a bucket unless it contains the date range you are searching for. Once the machine determines that a bucket contains the date range, new versions of Splunk can use Bloom filters [2] to perform a second check to see whether the query term may be in that bucket. Bloom filters provide Splunk with a very small chunk of data that can be used to definitively say that the search term does NOT appear in the bucket. A Bloom filter is 128 K per bucket, and because it's so small, it's fast to search and is likely to be cached in RAM. Bloom filters cannot say that the data IS in the bucket; that requires a full index search.

Inside each bucket, Splunk keeps the raw logs in a set of gzipped files and has a large amount of index data—this time using index in its traditional database meaning of the term—data that makes it faster to search than retrieving the raw log data. This index data can easily be the majority of the storage space. I have observed that with a bucket size of 10 GB, 3–4 GB is the gzipped raw log data (holding ~30 GB of raw data), and the remaining 6–7 GB is index data. Other users have reported even worse ratios: 2 GB of compressed data (13 GB raw) with 8 GB of index data. The Splunk documentation gives a rough estimate that your raw data will compress to ~10% its original size, and Splunk will use from 10% to 110% of the raw data size for indexes. For the rest of this article, I use Index to refer to the high-level structure and index when I need to talk about this lower level data. Luckily, only Splunk admins need to deal with indexes, and even they don't need to do much with them. Unlike indexes in a traditional database, Splunk indexes require very little configuration.

Buckets go through a series of states, with size, time, count, and manual triggers to move a bucket from one state to the next.

Bucket States

HOT

- Writable: all new data that arrives is written to a Hot bucket.
- If there are multiple Hot buckets, Splunk will try to put logs with different timestamps in different buckets to minimize the range of timestamps held in each bucket. This is only effective if you have logs arriving at the same time with different timestamps.
- Roll to Warm.

WARM

- Read-only.
- May be replicated to multiple machines for resiliency.
- Must live on the same file system as Hot buckets.
- Roll to Cold.

COLD

- Same restrictions as Warm, except that they can live on a different file system.
- Roll to Frozen via an admin configurable cold2frozen script.

FROZEN

- Not searchable.
- The default cold2frozen throws away the index data from the bucket to reduce the storage requirements. Thawing such a bucket requires re-indexing the raw data.
- Not managed by Splunk. The cold2frozen script should move them outside of the directory tree that holds Cold and Thawed buckets.
- Do not need to be online. These can be moved to offline storage (tape, etc.).
- The Splunk admin runs a frozen2thawed script to copy the bucket to Thawed. If the cold2frozen script throws away the index data, the frozen2thawed script will need to re-index the raw data. This can take a noticeable amount of time.

THAWED

- Logically the equivalent of Cold.
- Must live on the same file system as Cold buckets.
- Splunk expects these buckets to appear and disappear dynamically.

By allowing the data to be stored on two different file systems, you can have a small amount of fast but expensive storage for your "hot" and "warm" buckets that contain your most recent data (which theoretically is more likely to be searched) and a much larger amount of cheaper storage to hold your older data in "cold" and "thawed" buckets.

The "read-only" status of the buckets isn't quite true. There are times when an admin should go in under the covers with some of the Splunk command-line tools and modify the buckets (e.g., to split buckets that have bad timestamp ranges or that you are splitting into two Indexes), but such work is not supported by the Splunk GUI and is frowned upon by Splunk Support unless

you are doing it under the direction of the Splunk Professional Services or Support teams. If you are using Splunk data replication, the buckets will be modified to indicate which machines that bucket exists on.

To optimize your Splunk cluster, you will need to balance the following goals to match your system, log volume, and query patterns. There is no one right way to tune Splunk.

Goal #1

Make your buckets as large as you can.

The larger the bucket, the more efficient it is to search for things in the bucket. Searching through the indexes within a bucket is a O(log(n)) task, so searching through two buckets of size N will take just about twice as long as searching through a single bucket of size 2*N.

Goal #2

Appropriately set the number of hot buckets per Index.

If you have data arriving from machines that are providing significantly different timestamps, mixing this data into a single hot bucket will make each bucket's timestamp range wider by the range of timestamps considered "now" by your systems. This means that anytime a search is done against this wider time frame, this bucket will need to be searched. If you have buckets that roll every three hours, but combine local timestamps from every time zone into the same bucket, you will have to search eight times as many buckets than if you had all the systems reporting the same time.

If you have a machine whose clock is wrong and is generating logs with wildly wrong timestamps, it can be FAR worse than this (think about a system that has its clock off by months or years). If you keep logs in Splunk for long time periods, and do searches against old data routinely (security forensics searches are a good example), a problem like this can hang around for a very long time. This is one of the times where it's worth looking at manipulating the bucket contents to either split the bad data out or correct it.

If you cannot reliably set your systems to a uniform time, look at the time frame that a single bucket covers and see if you would benefit from allowing more "hot" buckets so that the different time zones each end up in their own hot bucket. If one hot bucket covers 48 hours, and you have two adjacent time zones, it's probably not worth splitting. However, if one hot bucket covers less than an hour, and you have two offices in time zones 10 hours apart, it's a very large win to have two hot buckets.

Ideally, you should use UTC instead of local time on all systems so that you also avoid daylight savings-related issues. If you have systems running on local time in different time zones, there are many ways that local times (without the time zone info) can make their way into your logs. For example, the traditional syslog timestamp does not contain time-zone information. Logs generated by applications frequently embed local times in the log messages themselves.

If you frequently have systems generate logs with wildly wrong timestamps (systems that boot with clocks off by years), add an extra hot bucket to catch these stragglers.

Goal #3

Segregate your data so that Splunk can easily rule out the need to search a bucket because it is in a different Index or because it can be eliminated via Bloom filters.

If you have logs that are distinct enough from each other that it's unlikely that you will be searching in both sets of logs at the same time (e.g., Web server and router logs), put the different logs in different Indexes. You can still search both if you need to by specifying both Indexes in your query, but if you are only searching for one type of log there will be fewer buckets to search.

Similarly, if one application generates logs that look very different, and you are likely to be searching for something that does not appear in one of the subsets of logs, putting those logs in a different Index and searching it by default will be a win because the Bloom filters will quickly exclude most of the buckets that you don't care about, so the resulting search will have far fewer buckets to search.

Goal #4

Have the logs spread out evenly over as many indexer systems as you can so that no matter what time frame you are searching for, you can utilize all of the systems. If your data is not distributed evenly across the systems, the system that has the most data will take longer to do its portion of the search and will be the limiting factor in your performance.

Splunk scales horizontally very well. As a result, the more systems that you have working on a given query, the faster your results will be returned.

With the new data replication features introduced in Splunk 5, and new management features expected in Splunk 6, you can have one box index the data and then use the Splunk replication to spread the data across multiple systems, using all of the systems to perform searches on the Index. In Splunk 5, the management tools do not allow you to do this effectively.

The other approach is to spread the data across different Indexer machines as it arrives, with each machine processing a portion of the data for that time frame. If you are using the Splunk agent to deliver the logs, you can use its load balancing mechanism to spread the data. If you are using syslog to deliver the logs, you

can use any appropriate load balancing mechanism to spread the logs across the machines. Note that you do not need to try and be perfect here; even something as crude as sending several seconds' worth of logs to one machine, and then switching to the next machine for the next several seconds is going to be good enough in practice, because your search time frames are generally in minutes at the most precise and can be in years. As the load scales up, you will find that one machine cannot keep up with the data flow, and load balancing will let you use multiple machines. In this case, try to tune the bursts of traffic to each machine to be small enough that the entire burst gets handled in a reasonable time. It would not be reasonable to send all logs for a five-minute period to one machine if it takes that machine an hour to process those logs. However, it would probably be very reasonable to send all logs for a second to that machine if it only takes it 12 seconds to process the logs; it's almost certainly reasonable to send 1/100 of a second worth of logs to a machine and have them be processed in 0.12 seconds.

Remember that if the data is spread unevenly across the systems, you will end up waiting for the slowest system to finish. Think about failure conditions when you are planning this, and if you have an extended outage that causes the systems to become unbalanced, you may need to go in under the covers again to get them closer to balanced.

Goal #5

The number of indexes times the number of hot buckets per index times bucket size needs to not only fit in RAM, but leave enough RAM left over for your searches (both the state info needed for real-time searches, and the cache space needed to effectively retrieve data from disk for your searches).

Note that this goal is in direct conflict with the earlier ones. The earlier ones drive you toward more indexes, more hot buckets, and larger bucket sizes, whereas this goal is saying that you need to minimize the product of these three values.

Also, if you have hot buckets that do not receive new logs for an extended time frame, they may end up getting pushed out of cache by the OS, resulting in worse performance as they have to get pulled back in.

When you do a search, Splunk will have to do its search through the Bloom filter and indexes of that bucket. The searches through the indexes are random searches over a large volume of data. If you have very large volumes of data compared to the RAM available for disk caching on your systems (after you account for the OS, the hot buckets, and the real-time searches), you will not have this data cached and so will have to read it from disk. This is both a very common case (it's common to have a single Splunk node with multiple TBs of indexed data, and it's unusual to have more than a few tens of GB of RAM available for caching) and a worst case random read load for spinning drives. If the version of Splunk you are running allows it, and you can afford it, putting the Bloom filter files on SSD storage can go a long way towards mitigating the cost of finding and reading them in after they are pushed out of the systems' disk cache.

Because a new log arriving can result in changes to the bucket indexes that result in rewriting large portions of the index data, you *really* want to have all of your hot buckets in RAM at all times; if any of them get pushed out to disk, you are likely to have to retrieve the data and rewrite it in the near future.

Remember that you do not have to put logs for every Index on every machine; you can split the Splunk cluster into multiple sub-clusters, with each sub-cluster holding specific types of logs.

Goal #6

Minimize the number of buckets that need to be retrieved for each query.

This goal requires a lot more explanation because it is in direct conflict with the earlier goals.

If you commonly do searches across different Indexes (if you have one Index per application, but need to look for someone accessing any application, for example), consider combining the Indexes together. The resulting Index will be less efficient to search than either Index on its own, but it may be significantly faster to search the combined Index than the two separate Indexes for a single query.

In addition, because the bucket sizes are constant, you may find that the time frames you commonly search are poorly matched with the time ranges that the buckets end up containing.

For example, given four log sources, each generating the same amount of data, you put them each into a separate Index, and the bucket size combined with your data rate means that each bucket contains four hours' worth of logs. A search for something across all four log sources over the last hour will still have to search one bucket per Index for a total of four buckets.

If, however, you were to combine all four log sources into a single Index, then each bucket will contain one hour's worth of logs, and a search over the last hour will only have to search a total of one bucket.

On the other hand, if your typical search is instead limited to a single log source, but it covers the last day, putting each application in its own Index will require searching six buckets, while having all four applications in the same Index will require searching 24 buckets.

Unless you are really sure that you have people doing a lot of searches of a specific set of data, I suggest starting out with everything in one Index and splitting it out later as you discover the true query patterns of your users.

Even if you have people interested in only a specific set of data, if they don't bother restricting their search to a particular Index, their query will be evaluated against all the indexes that their user permissions default to.

Special Case: Summary Log Data

Summary log data is a special case. Because it's fairly small and, as a result, queries against a time-range of summary data tends to want to retrieve all the data, it can be a good idea to have a separate Index for your summary data. Instead of setting the bucket size for the summary data Index, set the maxHotSpanSecs parameter to either 86400 (1 day) or 3600 (1 hour), and have Splunk rotate the hot buckets every hour every day at midnight or on the hour. The buckets are smaller, so a report over a massive time range will be a little less efficient than a large bucket, but the smaller bucket sizes match the time limits of your queries nicely, and it's much better to search for an hour's worth of data in a one-day bucket than in a three-week bucket. Summary data is also going to be searched far more frequently than normal data (largely due to dashboards).

As a result of this, and the fact that the data tends to be small, in many cases it makes sense to set up two separate sets of servers: one to handle your summary data, the other to handle your main data. The servers you use to handle the summary data do not need to have the super-beefy disk systems that your main data systems have; it's likely that you can set up systems that will keep the working set of the data that your users are querying in RAM. At that point, CPU will become your limiting factor instead of the disk I/O that is normally the first limitation. By setting up a separate set of systems to serve the summary data to dashboard users, you ensure that the dashboards are not going to impact the performance of your main systems: the worst they will do is slow each other down as well as slow report generation.

Conclusion

Splunk can be a wonderful tool for exploring your logs. It works very well for a wide range of user expertise. But if you don't have someone with a high level of expertise managing the system (very similar to the way that you would not dream of running a large Oracle system without a good DBA), you are likely to end up with a very poorly performing system. I've seen a Splunk cluster configured such that it would have required 40 TBs of RAM to cache all the hot buckets Splunk was trying to manage. I've also seen Splunk get to the point where the dashboards were showing data over three hours old because of poor tuning and bad usage patterns.

I am not saying that you need to prevent people from using Splunk, even for dashboards and reports. However, you do need to avoid giving everyone admin rights to Splunk, and you need to have a team of experts monitor and configure Splunk and how it's used so that you can maintain good performance of your system. If you don't do this, you are going to either end up building a cluster that is huge and expensive compared to what it really needs to be, or have it slow to a crawl.

This article is the latest in this series on logging topics [3]. If you have requests for topics for future articles, please email me (david@lang.hm) or Rik Farrow (rik@usenix.org) with your suggestions.

References

[1] "Logging Reports and Dashboards," *;login:*, vol. 39, no. 1, February 2014: https://www.usenix.org/publications/login/feb14/logging-reports-dashboards.

[2] http://docs.splunk.com/Documentation/Splunk/5.0.2/indexer/Bloomfilters.

[3] http://www.usenix.org/login/david-lang-series.