Revealing the Magic

The Lifecycle of a Splunk Search

Kellen Green  |  Senior Software Engineer

September 27th, 2017  |  Washington, DC
About Myself
web developer
<table>
<thead>
<tr>
<th>i</th>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1/31/17 11:24:50.000 PM</td>
<td>{ [-] bar: 24 foo: 90 time: 2017-01-31T23:24:50 +0000 }</td>
</tr>
<tr>
<td></td>
<td>1/31/17 10:43:30.000 PM</td>
<td>{ [-] bar: 2 foo: 95 time: 2017-01-31T23:43:30 +0000 }</td>
</tr>
</tbody>
</table>
“OUR DEVELOPERS HAVE PRODUCTIVITY SUPERPOWERS.”

Kris Wehner, VP of Engineering, Yelp Reservations
1. Develop a deeper understanding of the core components that make up a Splunk search.

2. Increase performance of your searches through more efficient queries.

3. Obtain stronger grasp of which deployment types are better suited for specific workloads.

Let's debunk that!
Data Set
26 event CSV file

- One event per day from Sept. 1 - 26
  - Random hour of the day

- Indexed field `foo`
  - Descending A - Z

- Unindexed field `bar`
  - Ascending Z - A
Search #1
Indexer Workflow

index="conf2017" foo="0"

Sept. 1\textsuperscript{st} to the 27\textsuperscript{th}
No Results?
index="conf2017" foo="0"

Yep, but I promise it's interesting!
Client to Indexer

index="conf2017" foo="0"
Indexes Directory
index="conf2017" foo="0"

▶ Root directory for indexes.
▶ Check if queried index directory exists.
▶ Specify an index to improve search performance.

$ cd $SPK_IDX/var/lib/splunk/
$ ls -l
  audit
  authDb
  conf2015
  conf2016
  conf2017
  defaultdb
  historydb
  Kvstore
Index Directory
index="conf2017" foo="0"

▶ colddb houses older searchable data.
  • Implement cheaper storage solutions.

▶ db directory for fresh data in high demand.

▶ Configurable in indexes.conf.

$ cd conf2017/
$ ls -l
- colddb
- datamodel_summary
- db
- thaweddb
Hot buckets are still being written to.

Warm buckets are event immutable.
  • Named by time range.

Specify strict time range to boost Performance.
Bloom Filter

index="conf2017" foo="0"

- Scanning buckets can be expensive.
- Bloom filter provides us with a fast way to determine if a term is NOT in a bucket.

```
$ cd db_1485388800_1493228720_1/
$ ls -l
        1485388800-1483228800.tsidx
bloomfilter
bucket_info.csv
Hosts.data
optimize.result
rawdata
Sources.data
SourceTypes.data
Strings.data
```
Bloom Filter Hashing

index="conf2017" foo="0"

\[
\begin{align*}
\text{hash}_1(\text{"foo=0"}) & = 0 \\
\text{hash}_2(\text{"foo=0"}) & = 7
\end{align*}
\]

- Second hash result points to \text{false}, so bucket will not contain matching events.
For search terms that are common, the bloom filter will do nothing to improve search performance.

Huge performance boost for rare and nonexistent events.
- Speed up on the order of 100x (1-2s to 10ms).
Search #2
Indexing

index="conf2017" foo="a"

vs

index="conf2017" bar="z"
Both Give Same Result

foo="a" vs bar="z"

2017-09-01T16:00:00 +0000,a,z
TSIDX File

foo="a" vs bar="z"

- Index file used to reduce the number of matching events.
- Lexigraphically sorted array of all terms within the bucket.
- The flag for |delete is also set here.
The Lispy query is used when searching through TSIDX files.

Created by the Search Head at search time.

foo="a" becomes [foo::a] in Lispy.

This will match all events where foo equals exactly a.
Lispy for Unindexed Fields

foo="a" vs bar="z"

- bar="z" becomes \[z\] in Lispy.

- This will match all events that contain \(z\) anywhere within the event.

- This might seem counter intuitive, but there is a good reason for this behavior.
Post TSIDX Results

foo="a" vs bar="z"

[foo::a]

<table>
<thead>
<tr>
<th>Time</th>
<th>Foo</th>
<th>Bar</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-09-01T16:00:00 +0000</td>
<td>a</td>
<td>z</td>
</tr>
</tbody>
</table>

[z]

<table>
<thead>
<tr>
<th>Time</th>
<th>Foo</th>
<th>Bar</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-09-01T16:00:00 +0000</td>
<td>a</td>
<td>z</td>
</tr>
<tr>
<td>2017-09-26T07:00:00 +0000</td>
<td>z</td>
<td>a</td>
</tr>
</tbody>
</table>
This search has completed and has returned 1 results by scanning 2 events in 0.26 seconds
(SID: 1502366143.92) search.log

Execution costs

<table>
<thead>
<tr>
<th>Duration (seconds)</th>
<th>Component</th>
<th>Invocations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>command.fields</td>
<td>1</td>
</tr>
<tr>
<td>0.00</td>
<td>command.search</td>
<td>1</td>
</tr>
<tr>
<td>0.01</td>
<td>command.search.expand_search</td>
<td>1</td>
</tr>
<tr>
<td>0.00</td>
<td>command.search.index</td>
<td>2</td>
</tr>
</tbody>
</table>
This search has completed and has returned 1 results by scanning 1 events in 0.056 seconds
(SID: 1502365983.83)

<table>
<thead>
<tr>
<th>Duration (seconds)</th>
<th>Component</th>
<th>Invocations</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>command.fields</td>
<td>1</td>
</tr>
<tr>
<td>0.00</td>
<td>command.search</td>
<td>1</td>
</tr>
<tr>
<td>0.01</td>
<td>command.search.expand_search</td>
<td>1</td>
</tr>
<tr>
<td>0.00</td>
<td>command.search.index</td>
<td>2</td>
</tr>
</tbody>
</table>
Raw Data Extraction

foo="a" vs bar="z"

$ cd rawdata/
$ ls -l
  journal.gz
  slicemin.dat
  slicesv2.dat

- journal.gz compressed slices of raw events.
- slices.dat map from TSIDX to slice.
- Remaining unwanted events will be filtered during extraction.
Cons of Unindexed Fields

- Increased number of potential matching events coming out of TSIDX.

- This list is kept in memory, leading to increased memory usage.

- More events, leads to more CPU needed for Journal decompression.
Index Everything?

foo="a" vs bar="z"

- This can quickly explode the size of your TSIDX files.
  - Leading to slow queries across the board.

- Only recommended for fields who's key-val pair is important, AND has a value which frequently occurs in other fields.
  - For example the pair foo="a" is important and often searched.
  - But bar="a", baz="a", and biz="a" are also common occurrences.
  - Then foo might make for a good index candidate.
Walklex Command
foo="a" vs bar="z"

$ walklex my.tsidx "foo::a"
0035130149.tsidx "foo::a"
my needle: foo::a
209 1 foo::a

$ walklex my.tsidx "z"
0035130149.tsidx "z"
my needle: z
287 2 z

- Shows us the number of matching TSIDX events for a given Lispy query.
- Useful for hunting down field indexing candidates.
Search #3

Wildcards

index="conf2017" foo="*a"

vs

index="conf2017" foo="a*"
Again Same Result
foo="*a" vs foo="a*"

2017-09-01T16:00:00 +0000,a,z
This search has completed and has returned 1 results by scanning 1 events in 0.056 seconds

(SID: 1502365983.83) search.log

| Execution costs |
|-----------------|-----------------|-----------------|
| Duration (seconds) | Component | Invocations |
| 0.00 | command.fields | 1 |
| 0.00 | command.search | 1 |
| 0.01 | command.search.expand_search | 1 |
| 0.00 | command.search.index | 2 |
Trailing Wildcard

foo="*a" vs foo="a*"

- Terms are sorted lexicographically within the TSIDX file.
- Binary search the index for the first matching term.
- For foo="*a", continue downward until we come to the first none matching term.
Leading Wildcard

`foo="*a" vs foo="a*"

- Same as trailing wildcard, start with the first matching term.

- However this time we must check all events that match our field name.

- Only when we get to "g", can we stop the search.
Trailing Wildcard + Unindexed
goo="*a" vs goo="a*"

▶ What if we searched for bar="*z"?

▶ Lispy is for this search is "[ ]".

▶ Skips TSIDX reducing altogether, relying completely on Journal extraction.
Search #4
Transactions

index="conf2017"

| transaction date_hour

vs

index="conf2017"

| stats count by date_hour
New Search

```
index=conf2017 | stats count by date_hour | chart count by date_hour
```

Events (20) Patterns Statistics (18) Visualization

Area Chart Format Line Ticks

![Graph](image)

<table>
<thead>
<tr>
<th>date_hour</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>11</td>
<td>2</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>2</td>
</tr>
<tr>
<td>21</td>
<td>2</td>
</tr>
<tr>
<td>22</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>2</td>
</tr>
</tbody>
</table>
Back to the Search Head

transaction vs stats
Directory of all saved and running searches on the Search Head.

sid can be obtained in the Job Inspector.

$ cd $SPK_SH/var/run/splunk/dispatch
$ ls -l
 1501601198.142
 1501601202.143
 1501601739.144
 1501601740.145
 1501601741.146
 1501601742.147
Search Folder
transaction vs stats

$ cd 1501601741.146/
$ ls -l
  args.txt
  buckets
  custom_prop.csv
  events
  timeline.csv
  info.csv
  peers.csv
  results.csv.gz
  search.log

- Collection of all data being returned from the indexers.

- results.csv.gz compressed events.

- timeline.csv UI timeline numbers.
### List View

<table>
<thead>
<tr>
<th>i</th>
<th>Time</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/31/17 11:24:50.000 PM</td>
<td>{ bar: 24, foo: 90, time: 2017-01-31T23:24:50 +0000 }</td>
<td></td>
</tr>
<tr>
<td>1/31/17 10:43:30.000 PM</td>
<td>{ bar: 2, foo: 95, time: 2017-01-31T18:43:30 +0000 }</td>
<td></td>
</tr>
</tbody>
</table>

### Timeline View

- File: `timeline.csv`
- File: `results.csv.gz`
Transaction Workflow

transaction vs stats
Stats Workflow

transaction vs stats
Performance boost for transactions running in parallel.
Distributed Search & Index Cluster

transaction vs stats

Scalable performance boost to stats and eval.
Stats Computation
transaction vs stats

- **stats** only concerns itself with a single event at once.

- Requires only one pass to complete the computation.

- For **stats count** Splunk returns value plus event occurrence count.
  - For example: hour "09" has 2 events.

<table>
<thead>
<tr>
<th>Time</th>
<th>Foo</th>
<th>Bar</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017-09-01T09:00:00</td>
<td>a</td>
<td>z</td>
</tr>
<tr>
<td>2017-09-02T07:00:00</td>
<td>b</td>
<td>y</td>
</tr>
<tr>
<td>2017-09-03T02:00:00</td>
<td>c</td>
<td>x</td>
</tr>
<tr>
<td>2017-09-04T13:00:00</td>
<td>d</td>
<td>w</td>
</tr>
<tr>
<td>2017-09-05T23:00:00</td>
<td>e</td>
<td>v</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2017-09-21T15:00:00</td>
<td>v</td>
<td>e</td>
</tr>
<tr>
<td>2017-09-23T16:00:00</td>
<td>w</td>
<td>d</td>
</tr>
<tr>
<td>2017-09-24T09:00:00</td>
<td>x</td>
<td>c</td>
</tr>
<tr>
<td>2017-09-25T06:00:00</td>
<td>y</td>
<td>b</td>
</tr>
<tr>
<td>2017-09-26T10:00:00</td>
<td>z</td>
<td>a</td>
</tr>
</tbody>
</table>
Splunk must iterate over each event for every transaction window.

Looking at a time complexity difference between $n$ and $n^2$.

Running only on a single Search Head doesn't help the situation.
Search #5
transaction plus stats

index=conf2017
| transaction foo
| stats count by foo
Where Does it Run?

transaction plus stats

- Splunk runs everything on the Indexer, until the first "slow" command forces otherwise.

- Everything trailing that command, will be forced to run on the Search Head.

- transactions and joins are examples of commands which would trigger this behavior.
<table>
<thead>
<tr>
<th>reduceSearch</th>
<th>transaction foo</th>
</tr>
</thead>
<tbody>
<tr>
<td>remoteSearch</td>
<td>litsearch index=conf2017</td>
</tr>
<tr>
<td>reportSearch</td>
<td>stats count by foo</td>
</tr>
<tr>
<td>request</td>
<td>{ [-]</td>
</tr>
<tr>
<td></td>
<td>adhoc_search_level: smart</td>
</tr>
<tr>
<td></td>
<td>auto_cancel: 30</td>
</tr>
<tr>
<td></td>
<td>check_risky_command: false</td>
</tr>
<tr>
<td></td>
<td>custom.dispatch.earliest_time: 0</td>
</tr>
<tr>
<td></td>
<td>custom.dispatch.latest_time:</td>
</tr>
<tr>
<td></td>
<td>custom.dispatch.sample_ratio: 1</td>
</tr>
<tr>
<td></td>
<td>custom.display.general.type: statistics</td>
</tr>
<tr>
<td></td>
<td>custom.display.page.search.mode: smart</td>
</tr>
<tr>
<td></td>
<td>custom.display.page.search.tab: statistics</td>
</tr>
<tr>
<td></td>
<td>custom.search: index=conf2017</td>
</tr>
<tr>
<td></td>
<td>earliest_time: 0</td>
</tr>
<tr>
<td></td>
<td>indexedRealtime:</td>
</tr>
<tr>
<td></td>
<td>latest_time:</td>
</tr>
<tr>
<td></td>
<td>preview: 1</td>
</tr>
<tr>
<td></td>
<td>provenance: UI:Search</td>
</tr>
<tr>
<td></td>
<td>rf: *</td>
</tr>
<tr>
<td></td>
<td>sample_ratio: 1</td>
</tr>
</tbody>
</table>
1. Leverage `stats` and `eval` over transactions whenever possible.

2. Choose trailing wildcards over leading in queries that require such functionality.

3. Look into indexing important fields who shares values with other fields.

4. Move slow commands as far right into the query string as possible.

Takeaways

You're all wizards now!
No Magic
Just Splunk
Don't forget to rate this session in the .conf2017 mobile app
Q&A