.conf2015

Search Optimization

Duncan Turnbull
Sales Engineer, Splunk
Julian Harty
Sales Engineer, Splunk

splunk>

Disclaimer

During the course of this presentation, we may make forward looking statements regarding future events or the expected performance of the company. We caution you that such statements reflect our current expectations and estimates based on factors currently known to us and that actual events or results could differ materially. For important factors that may cause actual results to differ from those contained in our forward-looking statements, please review our filings with the SEC. The forward-looking statements made in the this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, this presentation may not contain current or accurate information. We do not assume any obligation to update any forward looking statements we may make.

In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only and shall not, be incorporated into any contract or other commitment. Splunk undertakes no obligation either to develop the features or functionality described or to include any such feature or functionality in a future release.

Who Are We?

Julian Harty

- Sales Engineer
- US Based
- Recovering Oracle DBA
- Wantabee Aerobatic pilot
- Working on large scale deployments in the SF Bay Area

Duncan Turnbull

- Sales Engineer
- UK based
- Using Splunk since version 2.2.3
- Loves Lego!
- Working with large scale deployments
- Formerly doing PS and Training

Topics

- **Search Scoping:** A little background on Splunk Internals
- **Search Optimization tools:** SOS and Job inspector
- **Laying the groundwork for:** Regular Expression optimization
- **Beyond the basics:**
  - Joining Data
  - Transactions with Stats
  - Optimizing transaction
- **Bonus:**
  - Using tstats

Philosophy behind Search Optimization

- Don't feel the need to optimize every single search - focus on those which are frequently used and have the best potential for speedup. - KISS
- Understand the whole problem
- Know a small number of tricks well

How Can We Make Things Faster?

For all Searches:

- Change the physics (do something different)
- Reduce the amount of work done (optimize the pipeline)

In distributed Splunk environments particularly:

- How can we ensure as much work as possible is distributed?
- How can we ensure as little data as possible is moved?

.conf2015

The Basics – Search Scoping

splunk>

Time Range

- Splunk organizes events into buckets by time, which contain events
- The shorter the time range the fewer buckets will be read
- Common Practice: Searches running over all time
- Diagnostic: look for searches over all time

Time Range

- Good Practice: Scope to an appropriate shorter time range (using time range picker or earliest=/latest= or _index_earliest=/ _index_latest=)
- Speedup Metric: 30x – 365x
- Example: All Time -> Week to Date

ui-prefs.conf.spec [edit]

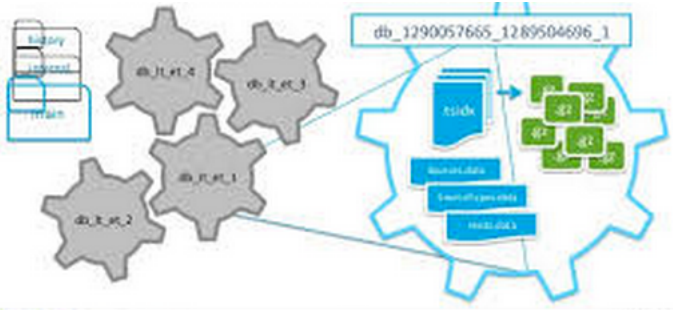
```
#   Version 6.2.5
#
# This file contains possible attribute/value pairs for ui preferences for a view.
#
# There is a default ui-prefs.conf in $SPLUNK_HOME/etc/system/default. To set custom
# configurations, place a ui-prefs.conf in $SPLUNK_HOME/etc/system/local/. To set cus
```

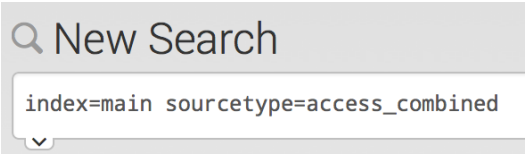
Presets

Real-time	Relative		Other
30 second window	Today	Last 15 minutes	All time
1 minute window	Week to date	Last 60 minutes	
5 minute window	Business week to date	Last 4 hours	
30 minute window	Month to date	Last 24 hours	
1 hour window	Year to date	Last 7 days	
24 hour window	Yesterday	Last 30 days	
All time (real-time)	Previous week	Last 90 days	
	Previous business week	Last year	
	Previous month		
	Previous year		

Scope on Metadata Fields

- Index is a special field, controlling which disk location will be read to get results
- All events in Splunk have sourcetype and source fields and including these will improve speed and precision
- Common Practise: Often roles include 'All-non internal indexes', no index or sourcetype specfier
- Diagnostic: look for searches without explicit index= clauses





Scope on Metadata Fields

- Good practice: include a specific index=, sourcetype= set of fields. If using multiple related sourcetypes, use eventtypes which also include a sourcetype scope
- Expected Speedup: 2x – 10x
- Example
  - Before : MID=*
  - After: index=cisco sourcetype=cisco:esa:textmail MID=*
  - Using Eventtypes: index=cisco eventtype=cisco_esa_email with (sourcetype="cisco:esa:textmail" OR sourcetype=cisco:esa:legacy) AND (MID OR ICID OR DCID)

Search Modes

- Splunk's search modes control Splunk's tendency to extract fields, with verbose being the most expansive and exploratory and fast being the least
- Diagnostic: request.custom.display.page.search.mode = verbose
- Common Practice: Verbose Mode left on after using
- Good Practice: Use Smart or Fast mode (dashboard searches do this automatically)
- Speedup Metric: 2x -5x

Inclusionary Search Terms

- Inclusionary search terms specify events to keep
- Exclusionary search terms specify events to remove
- Exclusions are appropriate for many use cases (interactive usage, exclusion of known errors, specificity)

Inclusionary Search Terms

- Diagnostic: Large scan numbers versus final events
- Good Practice: Mostly inclusionary terms, small or no exclusionary terms
- Speedup Metric: 2x -20x

```
index=main sourcetype=access* NOT action=purchase
```

```
index=main sourcetype=access* AND (action=addtocart OR action=view OR action=new)
```

Field Usage

- Define fields on segmented boundaries where possible
- Splunk will try to turn field=value into value, can be customized with fields.conf/segmentors.conf
- Diagnostic: check the base lispy in your search.log

Field Usage

- Good practise: Repeat field values as search terms if required, or use fields.conf
- Example:
  - Before: guid=942032a0-4fd3-11e5-acd9-0002a5d5c5
  - After: (index=server sourcetype=logins 942032a0-4fd3-11e5-acd9-0002a5d5c5 guid=942032a0-4fd3-11e5-acd9-0002a5d5c5) OR (index=client eventtype=client-login source=/var/log/client/942032a0-4fd3-11e5-acd9-0002a5d5c5)

.conf2015

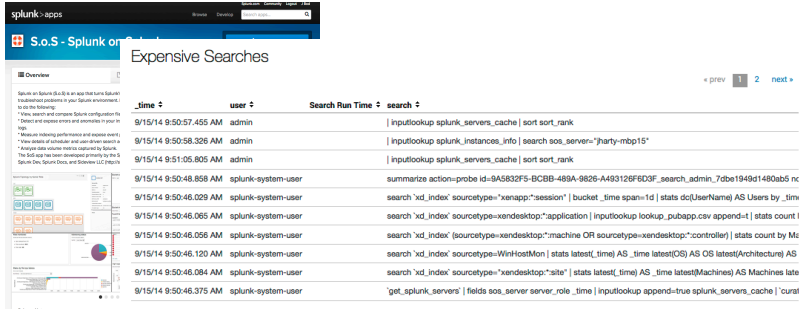
The Basics:
Search Scoping Tools

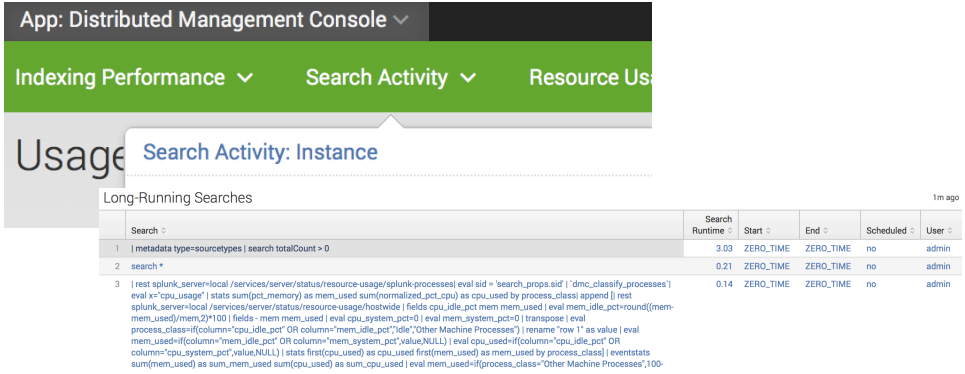
splunk>

A Word On Monitoring Searches

How do we easily identify less than optimal searches?

- SOS (Pre 6.1 Users)
- Distributed Management Console
- Job Inspector

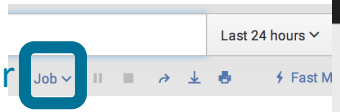




Measuring Search

Using the Splunk Search Inspector

Job Inspector



Search job inspector

This search has completed and has returned **1,178** results by scanning **1,783** events in **0.169** seconds.

The following messages were returned by the search subsystem:

DEBUG: **base lispy: [AND]**
DEBUG: **search context: user="tz", app="search", bs-pathname="/opt/customers/**

(SID: 1410925035.148)

Key Metrics:

- Completion Time
- Number of Events Scanned
- Search SID

Execution costs

Duration (seconds)	Component	Invocations	Input count	Output count
0.064	command.fields	67	437,295	437,295
70.029	command.remotetl	67	437,295	-
249.679	command.search	67	-	437,295
139.011	command.search.lookups	106	516,763	516,763
65.239	command.search.typer	67	437,295	437,295
13.921	command.search.tags	67	437,295	437,295
12.363	command.search.kv	106	-	-
11.834	command.search.rawdata	106	-	-
4.719	command.search.filter	106	-	-
2.264	command.search.index	327	-	-
0.109	command.search.fieldalias	106	516,763	516,763
0.407	dispatch.createProviderQueue	1	-	-
319.567	dispatch.stream.remote	57	-	15,711,327
83.738	dispatch.stream.remote.fool06.splunk.com	14	-	3,999,012
80.418	dispatch.stream.remote.fool04.splunk.com	13	-	3,932,267
78.599	dispatch.stream.remote.fool03.splunk.com	14	-	3,983,522
76.811	dispatch.stream.remote.fool05.splunk.com	15	-	3,776,581
0.001	dispatch.stream.remote.fool02.splunk.com	1	-	19,945
0.91	dispatch.timeline	73	-	-

Timings from the search command

Timings from distributed peers

Job Inspector Walkthrough – Search Command

Execution costs

Duration (seconds)	Component	Invocations	Input count	Output count
0.003	command.fields	13	189	189
0.035	command.lookup	13	189	189
0.015	command.remotetl	13	189	-
0.232	command.search	13	-	189
0.096	command.search.index	60	-	-
0.028	command.search.filter	4	-	-
0.004	command.search.calcfields	4	3,133	3,133
0.004	command.search.fieldalias	4	3,133	3,133
0	command.search.index.usec_1_8	74,656	-	-
0	command.search.index.usec_8_64	60	-	-
0.052	command.search.rawdata	4	-	-
0.045	command.search.kv	4	-	-
0.018	command.search.lookups	4	3,133	3,133
0.015	command.search.typer	13	189	189
0.013	command.search.tags	13	189	189
0.002	command.search.summary	13	-	-

Rawdata:
Improving I/O and CPU load

KV:
Are field extractions efficient

Lookups:
Used appropriately
Autolookups causing issues

Typer:
Inefficient Eventtypes

Alias:
Cascading alias

.conf2015

Laying the Groundwork

splunk>

Field Extractions

Most fields are extracted by regular expressions. Some regular expression operations are much better performing than others.

Field extractions can overlap – multiple TA's on the same source type for example.

Fields can also be from indexed extractions or structured search time parsing, as well as calculated (eval) fields and lookups

Duplicate Structured Fields

- Sometimes both indexed extractions and search time parsing are enabled for a CSV or JSON sourcetype. This is repeated unnecessary work, and confusing
- Diagnostic: duplicate data in multivalued fields
- Good Practice: Disable the search time KV
- Example:

```
[my_custom_indexed_csv]
# required on SH
KV_MODE=csv
# required on forwarder
INDEXED_EXTRACTIONS = CSV
```

```
[my_custom_indexed_csv]
# required on SH
KV_MODE=none
# required on forwarder
INDEXED_EXTRACTIONS = CSV
```

Basic Regular Expression Best Practice

- Backtracking is expensive
- Diagnostic: high kv time
- Good Practices:
  - Prefer + to *
  - Extract multiple fields together where they appear and are used together
  - Simple expressions are usually better (e.g. IP addresses)
  - Anchor cleanly
  - Test and benchmark for accuracy and speed

Basic Regular Expression Best Practice Examples

Before

- ’ (?P<messageid>[^]+)

After

- ^\S+\s+\d+\s+\d\d:\d\d:\d\d\s+\w+\[\d+\]\s+\w+\s+'\d+\.\d+\.\d+\s+(?P<messageid>[^]+)

Reading Job Inspector - search.kv

Execution costs

Duration (seconds)	Component	Invocations	Input count	Output count
0.014	command.fields	14	11,572	11,572
0.012	command.head	14	10,000	10,000
0.017	command.prehead	14	11,572	10,000
0.184	command.rex	14	10,000	10,000
0.634	command.search	28	11,572	23,144
0.037	command.search.index	14	-	-
0.029	command.search.filter	27	-	-
0.017	command.search.fieldalias	13	11,572	11,572
0.014	command.search.calcfields	13	11,572	11,572
0	command.search.index.usec_1_8	2,995	-	-
0	command.search.index.usec_8_64	672	-	-
0.285	command.search.lookups	13	11,572	11,572
0.159	command.search.kv	13	-	-

Search.KV=
Time taken to apply field extractions to events

How do you optimize this?
Regex optimizations

- Avoid Backtracking
- Use + over *
- Avoid greedy operators .*?
- Use of Anchors ^ $
- Non Capturing groups for repeats
- Test! Test! Test!

.conf2015

Beyond the Basics

splunk>

Lookups: Best Practice

- Use gzipped CSV for large lookups
- Add automatic lookups for commonly used fields
- Scope time based lookups cleanly
- Order lookup table by ‘key’ first then values
- When building lookups, use inputlookup and stats to combine (particularly useful for ‘tracker’ type lookups)
- Splunk will index large lookups

Reading Job Inspector - search.lookups

Execution costs

Duration (seconds)	Component	Invocations	Input count	Output count
0.011	command.fields	11	45,054	45,054
12.554	command.search	11	-	45,054
0.433	command.search.calcfields	28	45,054	45,054
0.145	command.search.fieldalias	28	45,054	45,054
0.031	command.search.index	33	-	-
0	command.search.index.usec_1_8	16	-	-
0	command.search.index.usec_8_64	14	-	-
6.663	command.search.lookups	28	45,054	45,054

Search.lookups =
Time to apply lookups to search

How do you optimize this?

- Use Appropriately (at end of search)
- Autolookups maybe causing issues

Joins: Overview

Splunk has a join function which is often used by people with two kinds of data that they wish to analyze together. It's often less efficient than alternative approaches.

- Join involves setting up a subsearch
- Join is going to join all the data from search a and search b, usually we only need a subset
- Join often requires all data to be brought back to the search head

Joins With Stats: Good Practice

- values(field_name) Is great
- range(_time) Is often a good duration
- dc(sourcetype) Is a good way of knowing if you actually joined multiple sources up or only have one part of your dataset
- eval Can be nested inside your stats expression
- searchmatch Is nice for ad-hoc grouping, could also use eventtypes if disciplined

Joins : Example

- Before:
  - Search A | fields TxnId,Queue | join TxnId [search B or C | stats min(_time) as start_time, max(_time) as end_time by TxnId | eval total_time = end_time - start_time] | table total_time,Queue
- After
  - A OR B OR C | stats values(Queue) as Queue range(_time) as duration by TxnId
- With more exact semantics:
  - A OR B OR C | stats values(Queue) as Queue range(eval(if(searchmatch("B OR C"), _time, null()))) as duration

Reading Job Inspector - search.join

Execution costs

Duration (seconds)	Component	Invocations	Input count	Output count
0.00	command.fields	3	208	208
0.01	command.join	4	208	208
0.03	command.search	6	208	416
0.00	command.search.filter	5	-	-

Search.join =
Time to apply join to search

How do you optimize this?

- Consider a dataset that is mostly error free and has a single unique identifier for related records
- Errors tie into the unique identifier
- Find the details of all errors
- Use a subsearch to first get a list of unique identifiers with errors:
- index=foo sourcetype=bar [search index=foo sourcetype=bar ERROR | top limit=0 id | fields id]

Using subsearch effectively

- Consider a dataset that is mostly error free and has a single unique identifier for related records
- Errors tie into the unique identifier
- Find the details of all errors
- Use a subsearch to first get a list of unique identifiers with errors:
- index=foo sourcetype=bar [search index=foo sourcetype=bar ERROR | top limit=0 id | fields id]

Reading Job Inspector - Subsearch Example

Execution costs

Duration (seconds)	Component	Invocations	Input count	Output count
0.048	command.fields	61	140,079	140,079
0.96	command.remotetl	61	140,079	-
1.633	command.search	122	140,079	280,158
0.229	command.search.filter	106	-	-
0.087	command.search.index	110	-	-
0.045	command.search.calcfields	45	140,079	140,079
0.045	command.search.fieldalias	45	140,079	140,079
0	command.search.index.usec_1_8	754	-	-
0	command.search.index.usec_8_64	79	-	-
0.958	command.search.rawdata	45	-	-
0.243	command.search.tags	61	140,079	140,079
0.052	command.search.kv	45	-	-

Search.rawdata =
Time to read actual events from rawdata files

How do you optimize this?

- Consider a dataset that is mostly error free and has a single unique identifier for related records
- Errors tie into the unique identifier
- Find the details of all errors
- Use a subsearch to first get a list of unique identifiers with errors:
- index=foo sourcetype=bar [search index=foo sourcetype=bar ERROR | top limit=0 id | fields id]

Key Items To Consider In Job Inspector

Job Inspector Conclusions: Search Command Summary

Component	Description
index	**look in tsidx files for where to read in rawdata**
rawdata	**read actual events from rawdata files**
kv	**apply fields to the events**
filter	filter out events that don't match (e.g., fields, phrases)
alias	rename fields according to props.conf
lookups	create new fields based on existing field values
typer	assign eventtypes to events
tags	assign tags to events

Job Inspector Conclusions: Distributed Search Summary

Metric	Description	Area to review
createProvider Queue	The time to connect to all search peers.	Peer conductivity
fetch	**The time spent waiting for or fetching events from search peers.**	**Faster Storage**
stream.remote	**The time spent executing the remote search in a distributed search environment, aggregated across all peers.**	
evaluate	The time spent parsing the search and setting up the data structures needed to run the search.	Possible bundle issues

Job Inspector / Search.log

FIeld	Description	Area to review
remoteSearch	The parallelizable portion of the search	Maximize the parallelizable part.
Base lispy / keywords	**The tokens used to read data from the index and events**	**Ensure contains field tokens**
eventSearch	The part of the search for selecting data	
reportSearch	The part of the search for processing data	

Worked Example

Stats vs Transaction

Search Goal: compute statistics on the duration of web session (JSESSIONID=unique identifier):

Not so Great:

```
> sourcetype=access_combined | transaction
JSESSIONID | chart count by duration
span=log2
```

Much Better:

```
> | stats range(_time) as duration by JSESSIONID
| chart count by duration span=log2
```

Use Stats To Maximal Effect

- Replace simple transaction or join usage with stats
- Stats count range(_time) dc(sourcetype) values(field) values(error) by unique_id
  - Gives you duration – range(_time)
  - Find incomplete 'transactions' with dc(sourcetype)
  - Find errors with values(error)
  - Find context with values(field)

Use Stats To Maximal Effect

- Consider using a base stats before expensive operations like eventstats or transaction or another stats:
  - | eval orig_time = _time | bucket _time span=1h| stats count range(orig_time) as duration by unique_id _time | eventstats avg(duration) as avg | where duration>avg

Reading Job Inspector - Stats Example

Execution costs

Duration (seconds)	Component	Invocations	Input count	Output count
0.048	command.fields	61	140,079	140,079
0.96	command.remotetl	61	140,079	-
1.633	command.search	122	140,079	280,158
0.229	command.search.filter	106	-	-
0.087	command.search.index	110	-	-
0.045	command.search.calcfields	45	140,079	140,079
0.045	command.search.fieldalias	45	140,079	140,079
0	command.search.index.usec_1_8	754	-	-
0	command.search.index.usec_8_64	79	-	-
0.958	command.search.rawdata	45	-	-
0.243	command.search.tags	61	140,079	140,079
0.052	command.search.kv	45	-	-

Search.rawdata =
Time to read actual events from rawdata files

How do you optimize this?

- Filtering as much as possible
- Add Peers
- Allocating more CPU, improving I/O

For More Info

- http://dunca.nturnbull.com/splunk/search
- http://docs.splunk.com/Documentation/Splunk/latest/Search/Writebettersearches
- http://docs.splunk.com/Documentation/Splunk/latest/Knowledge/ViewsearchjobpropertieswiththeJobInspector

Other Sessions To Look Out For:

- Smart Splunking - Jeff Champagne, Kate Engel
  - Tuesday 2:00 PM-12:45 PM

- Splunk Search Pro Tips - Dan Aiello
  - Wednesday 12:15 PM-1:00 PM

- Beyond the Lookup Glass: Stepping Beyond Basic Lookups – George Starcher, Duane Waddle
  - Check out the recording!

SPLing Bee Competition

Put your Splunk-fu to use in our first inaugural SPLing Bee!

- Your opportunity to learn new commands, show off your Splunk ninja skills and compete with your fellow Splunkers to solve Search challenges using Splunk!

When? Wednesday 4:15 PM-5:00 PM Breakout: 13

Bonus: Using tstats

- When using indexed extractions, data can be queried with tstats, allowing you to produce stats directly without a prior search
- Similarly data models can be queried with tstats (speedup on accelerated data models)
- Bonus: tstats is available against host source sourcetype and _time for all data (see also the metadata/metasearch command)
- Good Practice:
  - Use tstats directly for reporting searches where available
  - Read just the columns you need
  - Multiple queries usually better than a datacube style search

THANK YOU

Key Take away: Search Best Practice

Bad Behavior	Good Behavior	Performance Improvement	Comment
All Time Searches	-24h@h	365x 30x	Limit Time Range
>*	index=xyz source=www	10-50%	Index and default fields
> foo | search bar	> foo bar	30%	Combine Searches
Verbose Mode	Fast/Smart	20-50%	Fast Mode
A NOT B	A AND C AND D AND E	5-50%	Avoid NOTS
Searches over large datasets	Data Models and Report Acceleration	1000%	Use Intelligently
Searches over long periods	Summary Indexing	1000%	Use Sparingly