How to Scale: From _raw to tstats (and beyond!)

David Veuve

Staff Security Strategist, Splunk

.conf2016

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.

How to Use This Presentation

- This PDF is intended to be a reference guide, to complement the actual presentation.
- If you’ve already dabbled in tstats, feel free to read through. If you’re new to tstats, I highly recommend watching the video presentation first. I don’t do quite a good enough job with this slide verison for it to stand alone.
- Please find the video recording on the .conf website, maybe mid-to-late October (ask your Splunk team for updates if you don’t see it by then)

Agenda

1. Intro
2. David’s Story
3. Overview of Techniques (SI, RA, AP, tstats)
4. Data Models – What you need to know
5. How to transition from _raw to tstats
6. When Data Model Acceleration doesn’t work
7. Real World Examples
8. Advanced Topics

Personal Introduction

- **David Veuve** – Staff Security Strategist, Security Product Adoption
- SME for Architecture, Security, Analytics
- dveuve@splunk.com
- Former Splunk Customer (For 3 years, 3.x through 4.3)
- Primary author of Search Activity app
- Former Talks:
  - Security Ninjutsu Part Three: .conf 2016 (This year!)
  - Security Ninjutsu Part Two: .conf 2015
  - Security Ninjutsu Part One: .conf 2014
  - Passwords are for Chumps: .conf 2014

Why all this?

- Getting results fast is great, but only half the puzzle.
- If you / your team are writing searches that will run for 100 cpu hours per day, suppose that's 50% of your cluster's time.
- What if we could shrink that to 10 CPU hours? Your cluster just went from 75% utilized to 30% utilized.
- **Search acceleration lowers your TCO**
- **Search acceleration saves you time waiting**
- **Search acceleration lets you ask all of the questions**

Why this talk? Why now?

- tstats isn’t that hard, but we don’t have very much to help people make the transition.
- Everything that Splunk Inc does is powered by tstats.
- I’ve taught a lot of people in smaller groups about Search Acceleration technologies.
- To the masses!

Who are you?

- You are either a *super* hardcore dev, or you’re not brand new to Splunk.
- You’ve played with SPL. You understand how it works.
- You’re probably comfortable with stats.
- People probably come ask you for help building queries or solving problems.

What will you get?

- You’ll understand how to make queries that wow people.
- You’ll cement yourself as *the* office or user-group search ninja.
- You’ll happily learn how easy it is.

David’s Story

Just a boy, standing in front of a search command, asking it to show the syntax error.

Where I Started

- Customer at an advertising company
- Was a casual user, when I was handed a Business Analytics project
- Going from tens or hundreds of data points to millions
- Built tiered summary indexes
- Auto-switched between high granularity and low based on selected time windows
- Tons of help from Nick Mealy @ Sideview

Then I took a break

- I took two years off of Splunk, missing 5.x and the initial 6.0 release.
- Splunk released Report Acceleration
- Splunk released Data Model Acceleration

I Came to Splunk

- I rebuilt my dashboard. From Splunk 4 to Splunk 6, load time went from 1.5 min to 27 seconds.
- I used Report Acceleration – load time went down to 6 seconds

- But then I had a bunch of different searches running..

I Helped a Finance Company

- They wanted multiple dashboards, drilldown, searches, on 18 key fields in 2000 line XML documents.
- Built an accelerated data model with 18 calculated spath fields
- Used the pivot interface to build dashboards
- 30 day unaccelerated load time would have been **2 days** if I could wait
- 30 day accelerated load time was **15 seconds**

I Helped a Health Care Company

- They wanted distinct count of dest_ip per src_ip per day, averaged and stdev'd.
- Running over raw wasn't even considered.
- Depending on the analysis, we can search and process over **1 billion results / minute**.

Techniques

It’s all about the technique..

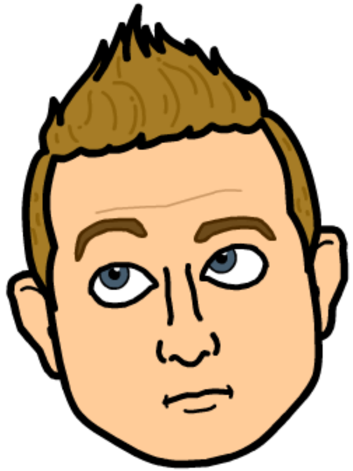
Summary Indexing

- Take the search you’re running right now, and store the results in a new index. No license required.
- How:
  - Just add | collect in your search, specifying destination index (maybe ”summary”)
  - Probably don’t want to use sistats, sitop, si..anything. They’re not really valuable.
  - http://www.davidveuve.com/tech/how-i-do-summary-indexing-in-splunk/
- Examples:
  - Store # of logins, # of distinct hosts, # of … per user / device / etc
  - Email logs are horrible and slow to process – store the output
  - ITSI Metric searches

Summary Indexing (2)

- Why: You're not accelerating raw events, you're accelerating the result of a search. We can't accelerate a search based datamodel. So: summary indexing

- Why not?
  - No Multiple Levels of Time Granularity --------->
  - Manual coordination of summary indexing ---->
  - Missed searches ------------------------------------->



Report Acceleration

- Takes a single saved search, with stats/timechart/top/chart and pre-computes the aggregates at multiple time buckets (per 10m, per hour, per day, etc., based on your acceleration range).
- Automatically switches between acceleration and raw data access when needed.
- You cannot query the data in ways that you didn't plan for originally

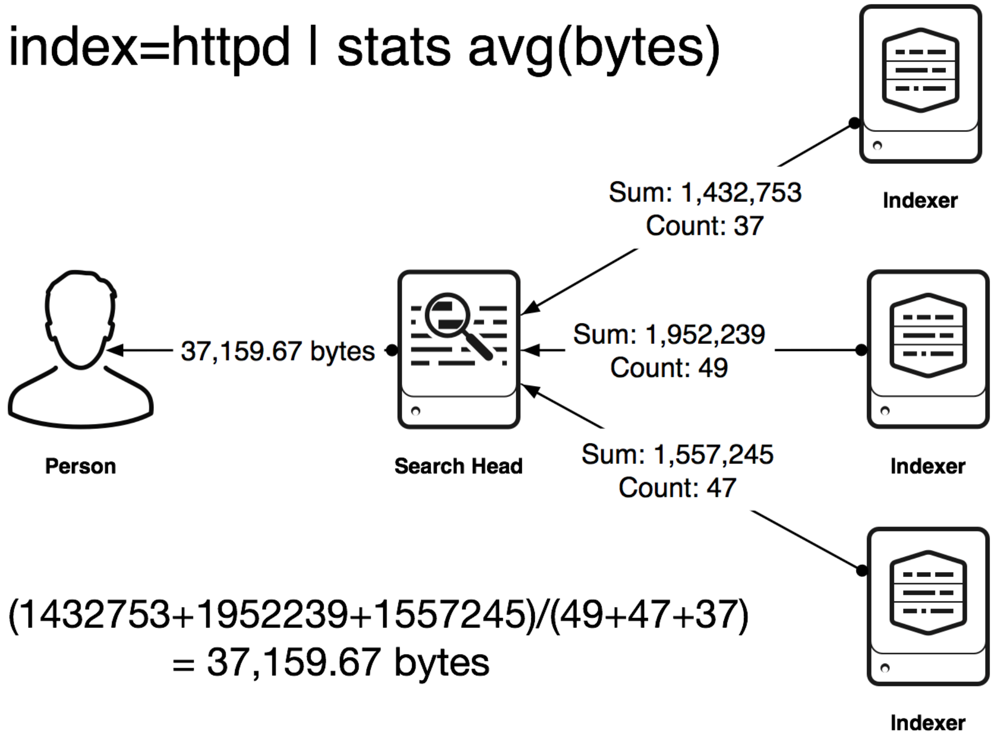
Report Acceleration (2)

- How:
  - Go into the saved search configuration and check the Accelerate box
  - Decide on over what time range you'd like to accelerate
  - Keep in mind that longer time ranges => less granularity (so if you choose 1 year, you'll lose 10min or 1 hr buckets
- Example
  - My exec dashboard needs to load, like, immediately.

Normal Search Example

- You ask for a statistical search
- Indexers return minimum necessary statistics (e.g., an avg needs sum / count)
- SH computes final result (sum(sum) / sum(count))

index=httpd | stats avg(bytes)

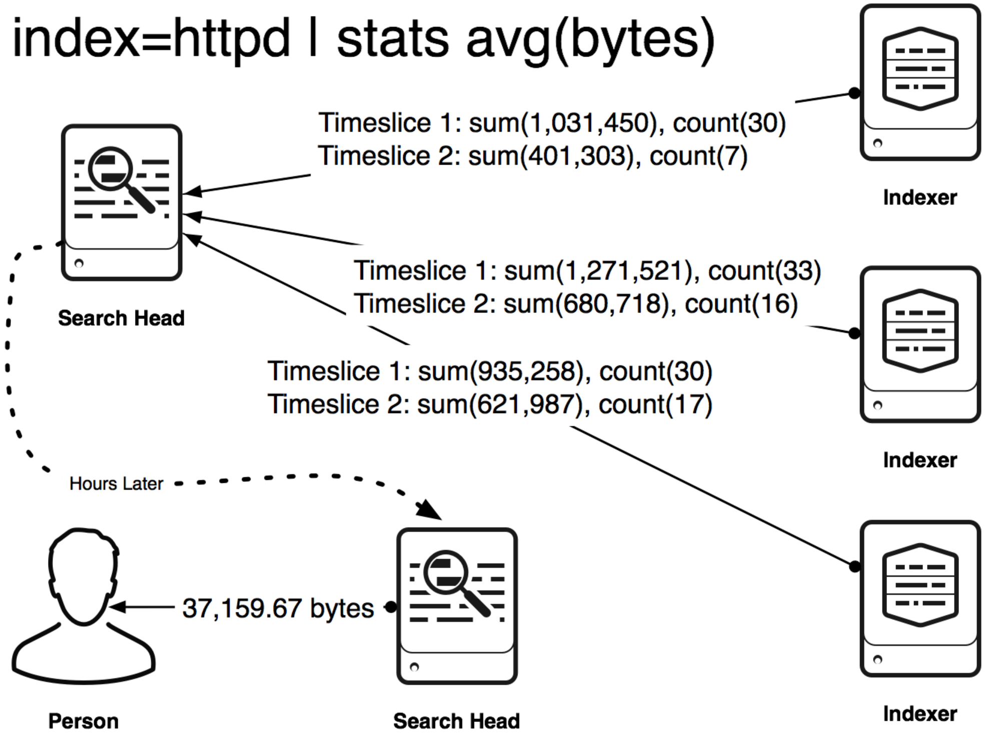


(1432753+1952239+1557245)/(49+47+37)
= 37,159.67 bytes

Report Acceleration Example

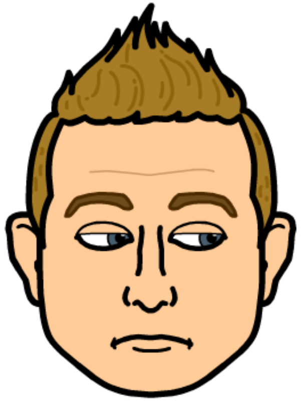
- SH regularly requests minimum necessary statistics (e.g., avg needs sum / count) split into time buckets
- Later, when user requests values, the SH already knows the answer.

index=httpd | stats avg(bytes)


Timeslice 1: sum(1,031,450), count(30)
Timeslice 2: sum(401,303), count(7)
Indexer
Search Head
Timeslice 1: sum(1,271,521), count(33)
Timeslice 2: sum(680,718), count(16)
Timeslice 1: sum(935,258), count(30)
Timeslice 2: sum(621,987), count(17)
Indexer
Hours Later
37,159.67 bytes
Indexer
Person
Search Head

Report Acceleration (3)

- Why?
  - You’ve got a small modest dataset with low split-by cardinality where you are willing to be crafty to run multiple queries
  - Auto fallback to raw logs, auto backfill and recovery, auto time granularity
  - SUPERFAST
  - Easy
- Why Not?
  - Mostly limited to a single search per job ------->
  - Only support for basic analytics ------------------>
  - Kinda a black art, not that widely used --------->

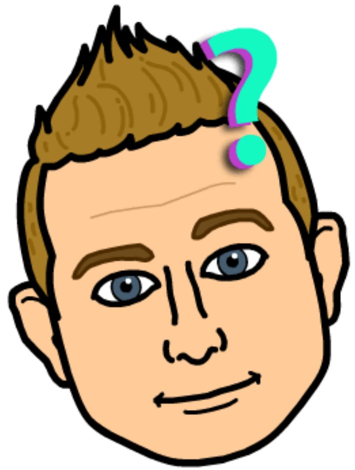


Accelerated Pivot

- Drag and drop basic stats interface, with the overwhelming power over accelerated data models on the back end
- How:
  - Build a data model (more on that later)
  - Accelerate it
  - Use the pivot interface
  - Save to dashboard and get promoted
- Examples
  - Your first foray into accelerated reporting
  - Anything that involves stats

Accelerated Pivot (2)

- Why?
  - Super easy
  - Automatically switch between raw logs and accelerated data
  - Data Model Acceleration = 💯
- Why Not?
  - Not entirely accelerated by default ---------------->
  - Can't go summariesonly in UI ----------------------->
  - Pivot search language is weirder than tstats ---->



tstats

- Operates on accelerated data models or tscollect files (and index-time field extractions, such as source, host, index, sourcetype, and those ITSI or occasional others)
- Can only do stats – no raw logs (today!)
- Is faster than you've ever imagined life to be.
- How:
  - Different search syntax, which takes adjustment, but actually really similar to normal stats.
  - | tstats count where index=* groupby index sourcetype
    - Bring a four-point seat harness 'cause we're going FAST

tstats (2)

- Why?
  - Distributed indexed field searching with the flexibility of search language to define syntax
  - summaries_only=t
  - Faster than you've ever been.

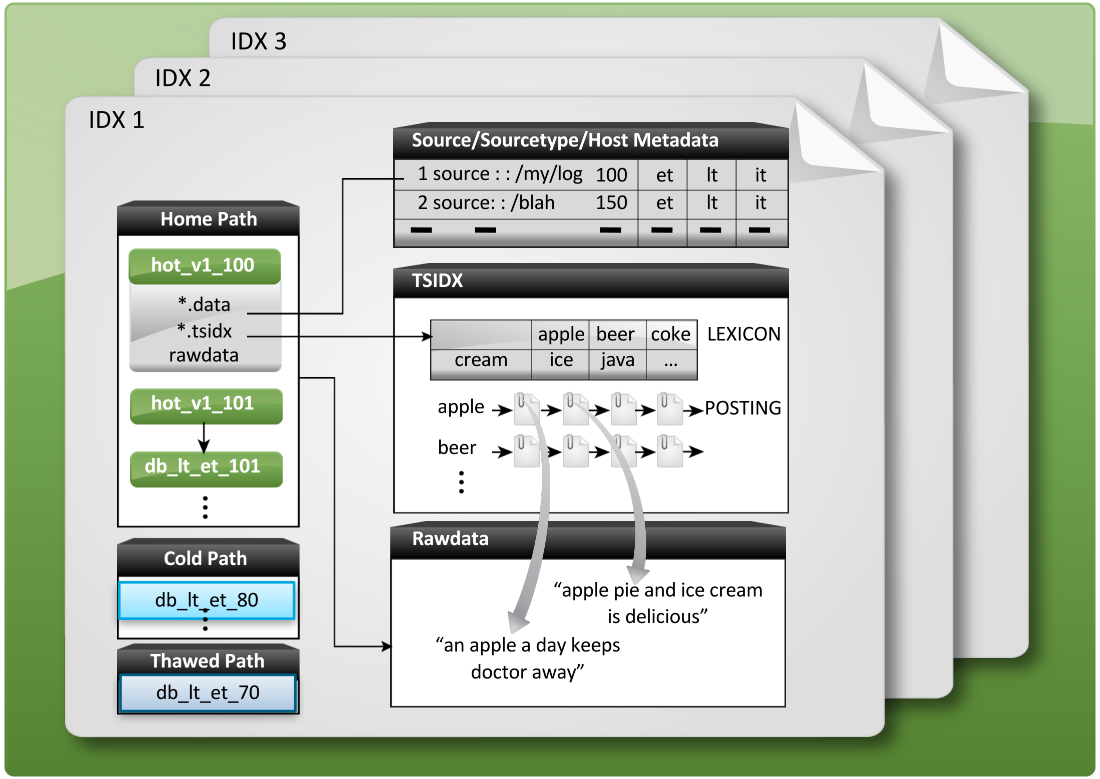
Data Models – What you need to know

Something clever here..

Data Model Basics

- Essentially anything you can define in props and transforms can go into an accelerated data model
- Only raw events – can't accelerate a data model based on searches, or with transaction, or etc.
  - Go check out summary indexing
- Favorite example: | eval myfield=spath(_raw, "path.to.my.field") is slow. Put that in your data model, and pivot/tstats queries will be superfast
- Next five slides from David Marquardt's .conf2013 Preso
  http://conf.splunk.com/session/2013/WN69801_WhatsNew_Splunk_DavidMarquardt_UnderstandingSplunkAccelerationTechnologies.pdf

Splunk Enterprise Index Structure



Raw data stored at offsets

Posting value	Seek address	_time	Raw events
0	42	1331667091	Deep likes Bud light
1	78	1331667091	Amrit likes Makers
2	120	1331667091	Ledion likes cognac
3	146	1331667091	Dave likes Jack Daniels
4	170	1331667091	Zhang likes vodka
5	212	1331667091	Deep likes Makers
6	240	1331667091	Dave likes Makers

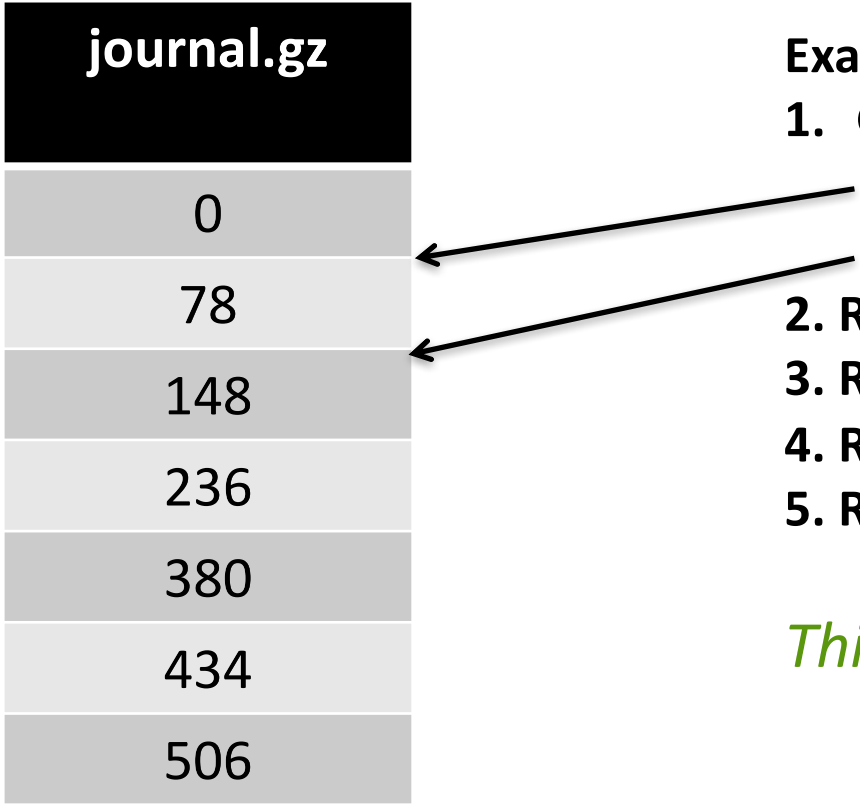
Raw Data Gets Indexed

- Each word in the raw event is indexed
- The TSIDX will store the offset #, and location in the gzip'd journal
- Querying dave makers returns #6

Raw events
Deep likes Bud light
Amrit likes Makers
Ledion likes cognac
Dave likes Jack Daniels
Zhang likes vodka
Deep likes Makers
Dave likes Makers

Term	Postings List
Amrit	1
Bud	0
Daniels	3
Dave	3,6
Deep	0,5
Jack	3
Ledion	2
Makers	1,5,6
Zhang	4
cognac	2
likes	0,1,2,3,4,5,6
light	0
vodka	4

Reading Compressed Rawdata



Example: Reading offsets (120, 170)

1. **Group offsets into residing chunks**
   - 120 falls into range (78, 148)
   - 170 falls into range (148, 236)
2. **Read data off disk and decompress**
3. **Run through field extractions**
4. **Recheck filters**
5. **Run calculations**

This is disk + CPU EXPENSIVE

Storing Indexed Fields in TSIDX

Term	Postings List
bar::AB	1,3,7,39,98
bar::cez	0,6,9,12
bar::xyz	3,4,5,6
baz::1	3,6,85
baz::2567	0,5
baz::462	3,24,45
baz::98	2,3,5,8,9
baz::99023	1,5,6,76,99
foo::afdjsi	4,567,2345
foo::aghdafo	2,234,6667
foo::bazcxuid	0,1,623,7777
foo::cef	0,1,2,3,4,43
foo::zaz	4

Big Idea: Use the lexicon as a field value store!

By simply separating fields and values with "::" we can store sufficient information to run more interesting queries.

Data Model queries **don't** ever visit raw logs. They live entirely within TSIDX!

How to Transition from _raw to tstats

A whole new world (don’t you dare close your eyes)

Process Overview

- Build your data model with whatever fields you could care about
- Start with your raw search
- Identify the aggregation that you want to do
  - Stats avg(bytes), dc(host), whatever else
- Make the minor syntax adjustments for tstats

Example Without Data Models

Raw

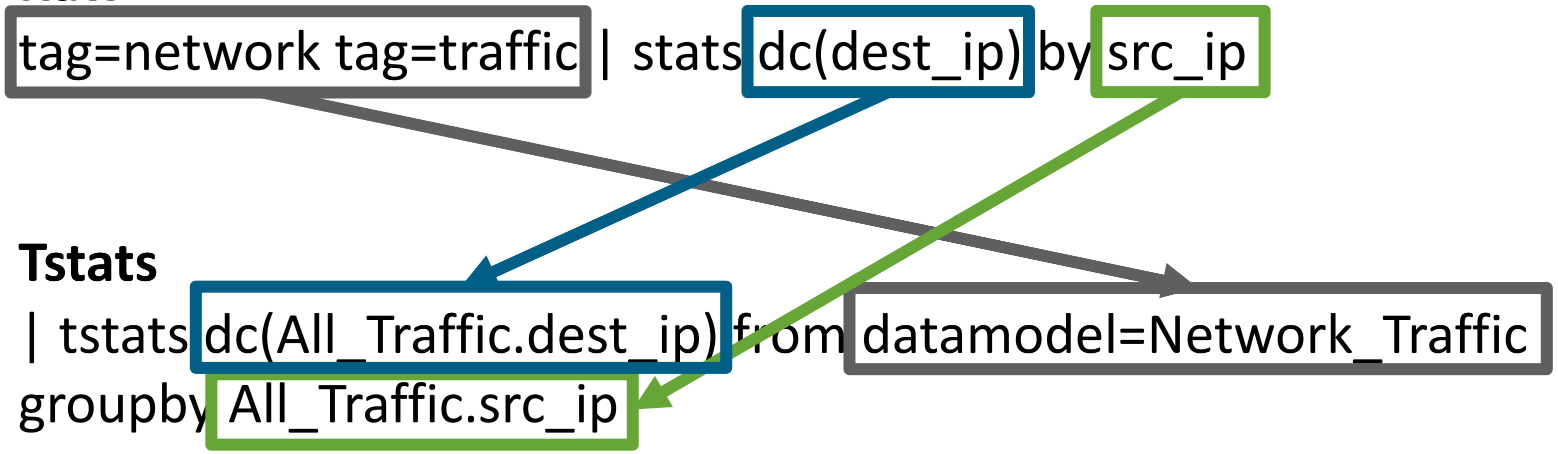
```
index=* | stats count by index, sourcetype
```

Tstats

```
| tstats count where index=* groupby index, sourcetype
```

Example With Data Models

Raw



```
tag=network tag=traffic | stats dc(dest_ip) by src_ip
```

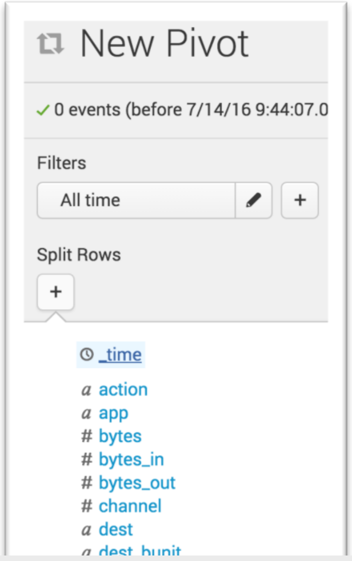
Tstats

```
| tstats dc(All_Traffic.dest_ip) from datamodel=Network_Traffic
groupby All_Traffic.src_ip
```

Challenge: Identifying Fields

- What fields are actually in a data model?
- How did I know to use “All_Traffic.dest_ip” instead of “dest_ip” or instead of “Network_Traffic.dest_ip?
- To figure it out, we can look at the data model definition via pivot, or at the resulting tsidx files via walklex
- Pivot doesn’t require SSH access, but still leaves you guessing for parts
- walklex is much more accurate and preferable

Identifying Fields via Pivot


New Pivot

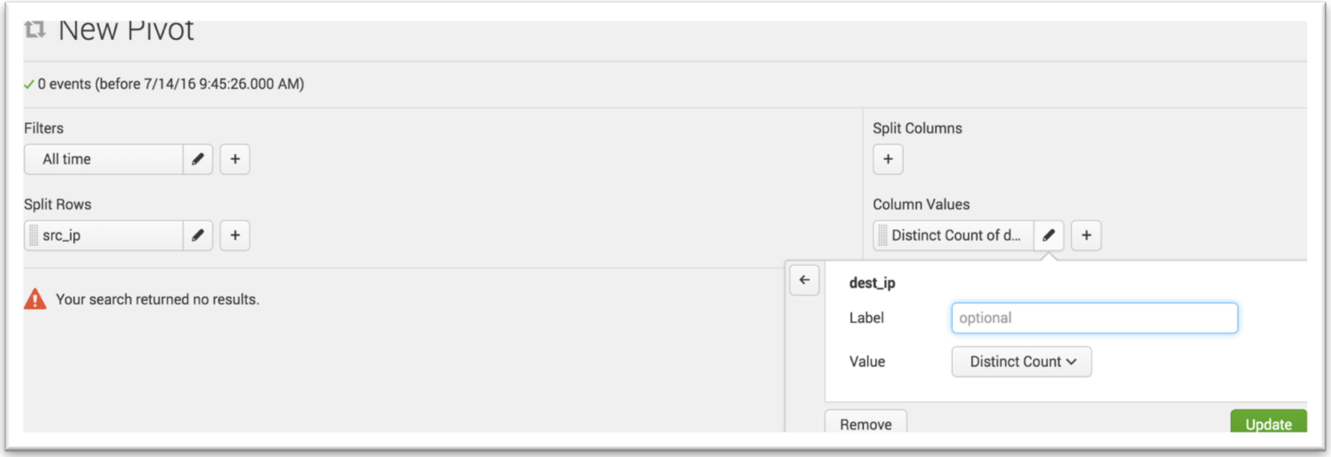
✓ 0 events (before 7/14/16 9:44:07.0

Filters

All time

Split Rows

_time
a action
a app
# bytes
# bytes_in
# bytes_out
# channel
a dest
a dest_bunit


New Pivot

✓ 0 events (before 7/14/16 9:45:26.000 AM)

Filters: All time

Split Rows: src_ip

Split Columns

Column Values: Distinct Count of d...

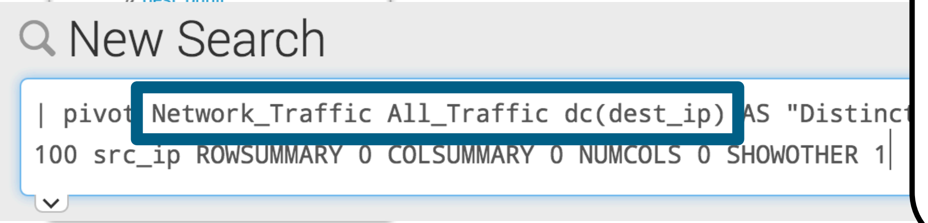
Your search returned no results.

dest_ip

Label: optional

Value: Distinct Count

Remove · Update


New Search

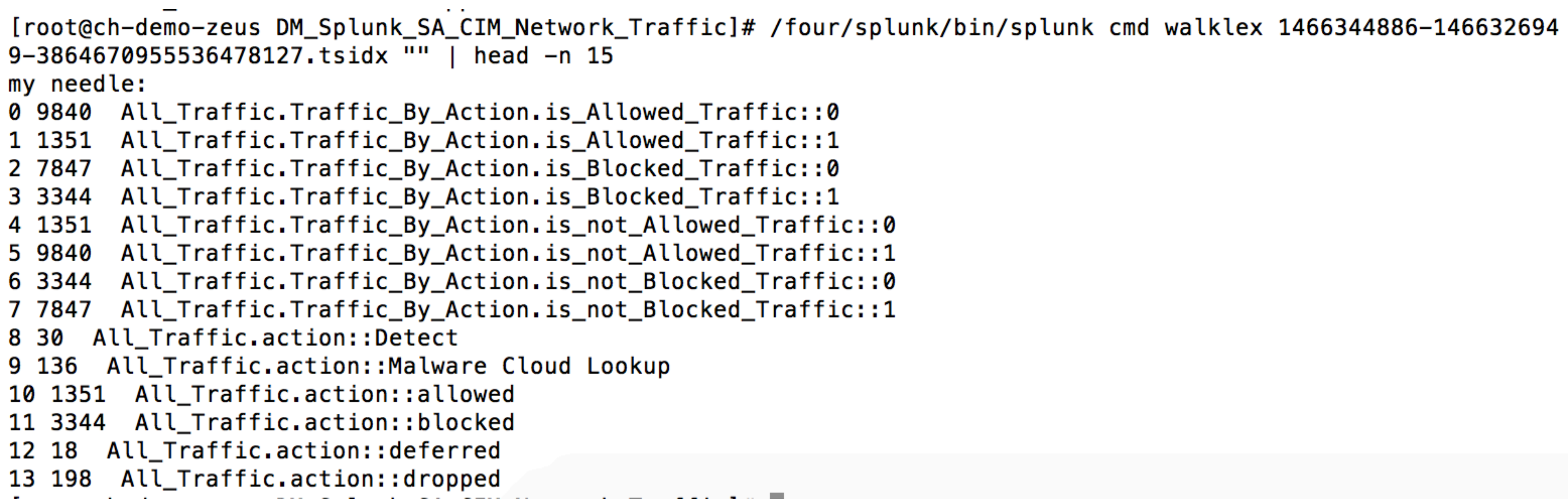
```
| pivot Network_Traffic All_Traffic dc(dest_ip) AS "Distinct
100 src_ip ROWSUMMARY 0 COLSUMMARY 0 NUMCOLS 0 SHOWOTHER 1
```

The data model is Network_Traffic, and the root event node is "All_Traffic" so fields should mostly be All_Traffic.fieldname

Identifying Fields via Walklex

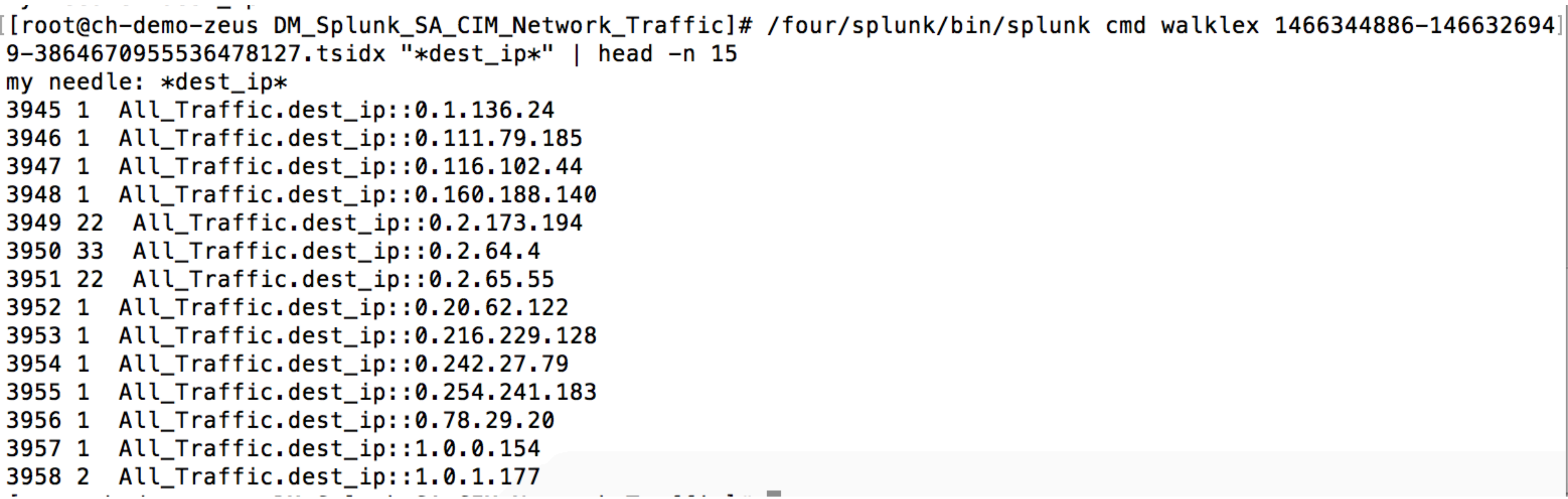
- Find the TSIDX File on your indexer (let's assume a data model)
  - Path set in your index config, but by default in the index folder
  - Usually $SPLUNK_HOME/var/lib/splunk/<INDEX>/datamodel_summary/<BUCKET_ID>/<SEARCH_HEAD_GUID>/<DATAMODEL_NAME>/<TIMERANGE>.tsidx
  - Good news: That's by far the hard part
  - Example: /opt/splunk/var/lib/splunk/defaultdb/datamodel_summary/1772_813B72E7-6743-4F46-9DE6-536F78929EDD/813B72E7-6743-4F46-9DE6-536F78929EDD/DM_Splunk_SA_CIM_Network_Traffic/1466344886-1466326949-3864670955536478127.tsidx
- Run walklex, either with an empty string "" or a wildcard "*dest_ip*"
  - $SPLUNK_HOME/bin/splunk cmd walklex <TSIDXFILE> ""

Example Walklex



```
[root@ch-demo-zeus DM_Splunk_SA_CIM_Network_Traffic]# /four/splunk/bin/splunk cmd walklex 1466344886-146632694
9-3864670955536478127.tsidx "" | head -n 15
my needle:
0 9840  All_Traffic.Traffic_By_Action.is_Allowed_Traffic::0
1 1351  All_Traffic.Traffic_By_Action.is_Allowed_Traffic::1
2 7847  All_Traffic.Traffic_By_Action.is_Blocked_Traffic::0
3 3344  All_Traffic.Traffic_By_Action.is_Blocked_Traffic::1
4 1351  All_Traffic.Traffic_By_Action.is_not_Allowed_Traffic::0
5 9840  All_Traffic.Traffic_By_Action.is_not_Allowed_Traffic::1
6 3344  All_Traffic.Traffic_By_Action.is_not_Blocked_Traffic::0
7 7847  All_Traffic.Traffic_By_Action.is_not_Blocked_Traffic::1
8 30  All_Traffic.action::Detect
9 136  All_Traffic.action::Malware Cloud Lookup
10 1351  All_Traffic.action::allowed
11 3344  All_Traffic.action::blocked
12 18  All_Traffic.action::deferred
13 198  All_Traffic.action::dropped
```

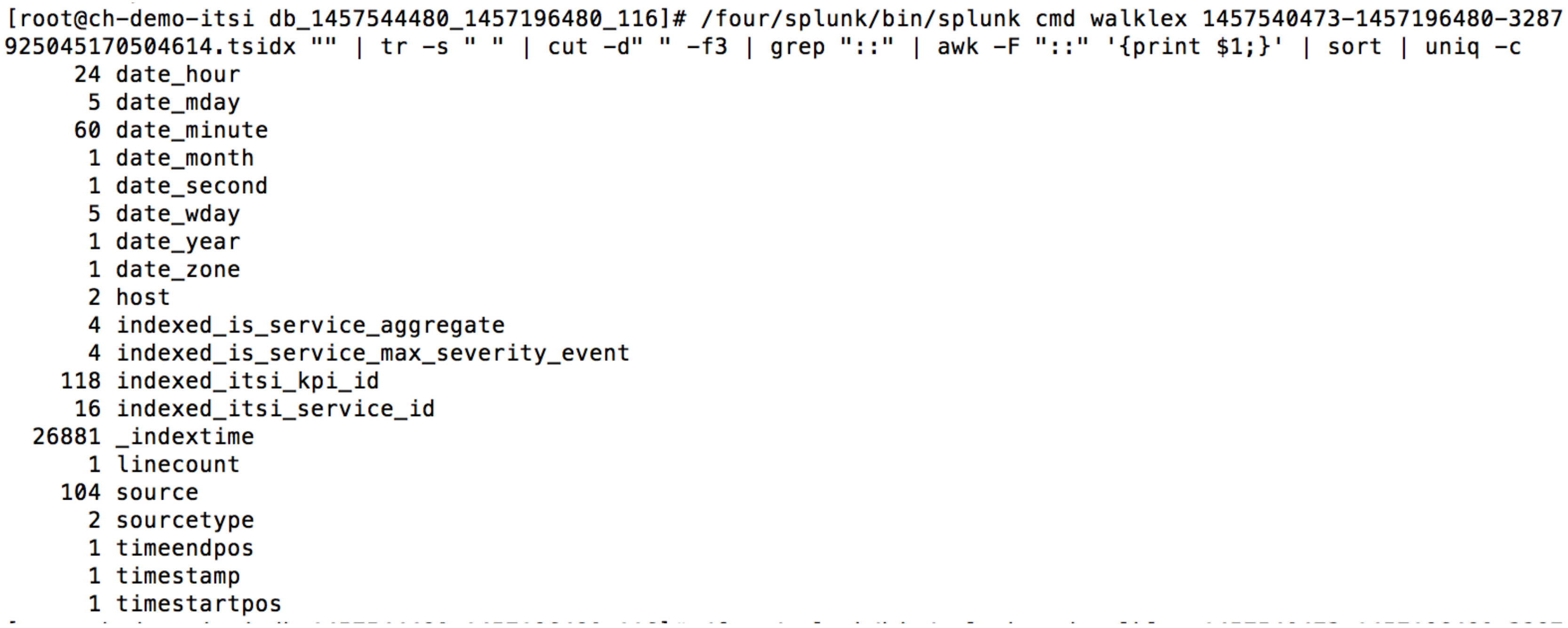
Example Walklex for a Particular Field



```
[root@ch-demo-zeus DM_Splunk_SA_CIM_Network_Traffic]# /four/splunk/bin/splunk cmd walklex 1466344886-14663269
9-3864670955536478127.tsidx "*dest_ip*" | head -n 15
my needle: *dest_ip*
3945 1  All_Traffic.dest_ip::0.1.136.24
3946 1  All_Traffic.dest_ip::0.111.79.185
3947 1  All_Traffic.dest_ip::0.116.102.44
3948 1  All_Traffic.dest_ip::0.160.188.140
3949 22  All_Traffic.dest_ip::0.2.173.194
3950 33  All_Traffic.dest_ip::0.2.64.4
3951 22  All_Traffic.dest_ip::0.2.65.55
3952 1  All_Traffic.dest_ip::0.20.62.122
3953 1  All_Traffic.dest_ip::0.216.229.128
3954 1  All_Traffic.dest_ip::0.242.27.79
3955 1  All_Traffic.dest_ip::0.254.241.183
3956 1  All_Traffic.dest_ip::0.78.29.20
3957 1  All_Traffic.dest_ip::1.0.0.154
3958 2  All_Traffic.dest_ip::1.0.1.177
```

Example Distinct Count of Walklex Fields

- /opt/splunk/bin/splunk cmd walklex 1457540473-1457196480-3287925045170504614.tsidx "" | tr -s " " | cut -d" " -f3 | grep "::" | awk -F "::" '{print $1;}' | sort | uniq -c



```
[root@ch-demo-itsi db_1457544480_1457196480_116]# /four/splunk/bin/splunk cmd walklex 1457540473-1457196480-3287
925045170504614.tsidx "" | tr -s " " | cut -d" " -f3 | grep "::" | awk -F "::" '{print $1;}' | sort | uniq -c
     24 date_hour
      5 date_mday
     60 date_minute
      1 date_month
      1 date_second
      5 date_wday
      1 date_year
      1 date_zone
      2 host
      4 indexed_is_service_aggregate
      4 indexed_is_service_max_severity_event
    118 indexed_itsi_kpi_id
     16 indexed_itsi_service_id
  26881 _indextime
      1 linecount
    104 source
      2 sourcetype
      1 timeendpos
      1 timestamp
      1 timestartpos
```

tstats where clause

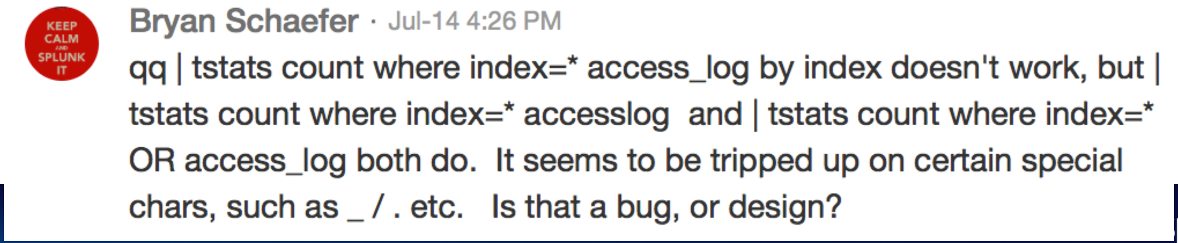
- Works surprisingly like the initial search criteria of a raw search
- where index=* sourcetype=pan_traffic OR sourcetype=pan:traffic
  - Just like normal search
- | tstats count where index=pan 10.1.1.1
  - With non-datamodel data, 10.1.1.1 will be in the tsidx.
- where earliest=-24h
  - Note that there is a bug in 6.3, 6.4 where a more restrictive timepicker range doesn't override the earliest=… (unlike in raw search – this is a bug)

tstats grouping by

- When grouping by values (e.g., src_ip, sourcetype, etc.) it’s like a normal stats …. by …
  - | tstats count where index=* **groupby source, index**
- You can also group by time, without using the bucket command
  - | tstats count where earliest=-24h index=* groupby index **_time span=1h**

Bugs and Surprises

- There's a bug in 6.3/6.4 with earliest and latest where tstats doesn't override the time picker, so easiest to leave your time picker at all time.
- Sometimes tstats handles where clauses in surprising ways. For example: no underscores in values, no splunk_server_group, no cidrmatches (All_Traffic.dest_ip!=172.16.1.0/24 – Fail. All_Traffic.dest_ip!=172.16.1.* – Success)



Bryan Schaefer · Jul-14 4:26 PM

qq | tstats count where index=* access_log by index doesn't work, but | tstats count where index=* accesslog and | tstats count where index=* OR access_log both do. It seems to be tripped up on certain special chars, such as _ / . etc. Is that a bug, or design?

When Data Model Acceleration or tstats Don’t Work

a sad, sad day….

On the output of a stats command

- Sadly, you can’t accelerate a search-based data model, so no luck.
- This is where Summary Indexing comes in
- You can also do index-time field extractions on summary indexes if you’re fancy, and then tstats on those!

Workaround: Stats -> SI + Index Time -> tstats

- Creating index time fields is a hassle, involving fields.conf, props.conf, transforms.conf, but it works on summary indexed data.
- For example, from ITSI, we index the field **indexed_itsi_kpi_id** from summary indexed searches (sourcetype: stash_new)

fields.conf:

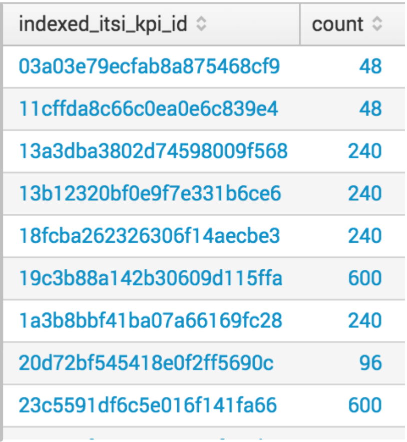
```
[indexed_itsi_kpi_id]
INDEXED=true
```

transforms.conf:

```
[set_kpisummary_kpiid]
REGEX = itsi_kpi_id\s*=\s*([^\s,]+)
WRITE_META = true
FORMAT = indexed_itsi_kpi_id::$1
```

props.conf:

```
[stash_new]
TRANSFORMS-set_kpisummary_index_fields = set_kpisummary_kpiid
```



indexed_itsi_kpi_id	count
03a03e79ecfab8a875468cf9	48
11cffda8c66c0ea0e6c839e4	48
13a3dba3802d74598009f568	240
13b12320bf0e9f7e331b6ce6	240
18fcba262326306f14aecbe3	240
19c3b88a142b30609d115ffa	600
1a3b8bbf41ba07a66169fc28	240
20d72bf545418e0f2ff5690c	96
23c5591df6c5e016f141fa66	600

When Your Cardinality is Crazy High

- Tstats can process huge numbers of events (billions, trillions, no problem).
- But if we have to store millions of rows in memory based on your split-by, that can be rough
- Example: 300,000 person company tracks # of logins per user per day over 100 days. 300,000 * 100 = 30M rows, which means writing partial results to disk and sadness.
- Better approach is to summary index each day, and then use tstats to process those results either via index-time summarization or DMA

When Any Cardinality is crazy crazy high

- tstats efficiency is fundamentally based on the assumption that a particular value will be used a few times.
- If you have millions of events, each with 10 data points, with 10 points of precision such that repeat values are unlikely, your tsidx file will be absolutely massive.

Real World Examples

When things stop being slow, and start getting real.

Splunk(x) - Index Searches

- For running our Splunk Internal UBA project, we needed to know what sourcetypes were in the system.
- _raw: index=* earliest=-24h | bucket _time span=1h | stats count by sourcetype, _time
  - Time to complete: 68,476 seconds (19 hours)
- tstats: | tstats count where index=* groupby sourcetype _time span=1h
  - Time to complete: 6.19 seconds
- Speed Difference: **11,062x** (not percent, eleven thousand times faster)
- Query Length difference: **18 characters shorter**

Financial Customer XML Use Case

- What Technology?
  - Accelerated Data Models with Pivot
- Why?
  - Heavy XML Parsing meant search queries were terribly slow
  - Pivot was very easy to use
- Result
  - Very high scale, very happy customer

Financial Customer XML Use Case (2)

- No XML extraction
- Raw: 8.811 seconds
  - index=xx-xxxx sourcetype=xxx-xxx splunk_server=myserver01.myserver.local ParticularLogIdentifier host=*ServerType* | timechart count by host
- Accelerated Pivot: 1.25 seconds
  - | pivot XXXXXXX YYYYYY count(YYYYYY) AS "Number of Events" SPLITROW _time AS _time PERIOD auto SPLITCOL host FILTER host is "*ServerType*" SORT 100 _time ROWSUMMARY 0 COLSUMMARY 0 NUMCOLS 100 SHOWOTHER 1
- Tstats Summaries Only: 0.896 seconds
  - | tstats summariesonly=t count from datamodel=XXXXXXX where (nodename = YYYYYY) (YYYYYY.host="*ServerType*") groupby _time
- Speed Difference: **9.9x Faster**
- Query Length: 18 characters shorter

Financial Customer XML Use Case (3)

- Single XML Extraction via spath
- _raw: 299.763 seconds
  - index=xx-xxxx sourcetype=xxx-xxx splunk_server=myserver01.myserver.local ParticularLogIdentifier | eval RuleId=spath(_raw, " ___path____.___to__._____very_______._____long___._:_xml___._:____._:_____.______.__________.__________.______________._ _____ ")| timechart count by RuleId
- Accelerated Pivot: 2.4 seconds
  - | pivot XXXXXXX YYYYYY count(YYYYYY) AS "Number of Events" SPLITROW _time AS _time PERIOD auto SPLITCOL RuleId SORT 100 _time ROWSUMMARY 0 COLSUMMARY 0 NUMCOLS 100 SHOWOTHER 1
- tstats summariesonly: 2.04 seconds
  - | tstats summariesonly=t count from datamodel=XXXXXXX where (nodename = YYYYYY) groupby RuleId_time
- Speed Difference: about **146.9x faster**
- Query Length: 50 characters shorter

Financial Customer XML Use Case (4)

- Heavy XML Extraction (mentioned earlier). Searches anonymized...
- An Entire Dashboard of Unaccelerated Pivots with lots of XML spath
  - Time to complete: 172,800 seconds (2 days)
- An Entire Dashboard of Accelerated Pivots
  - Time to complete: 16 seconds
- Speed Difference: about **10000x**
- Time Taken to Build 14 Panel Dashboard via Pivot: 15 minutes

ES Endpoint + Proxy + AV

- What Technology?
  - ES Data Models + tstats
- Why?
  - ES Data Models were already built, and multiple data sources so tstats append=t
- Result
  - Super fast search, high scalable.
  - Data Models make things easier
- Downside
  - In this case, a 19 second savings every 15 minutes = a $211 ROI/year on a $300k Splunk infrastructure… maybe not enough?

ES Endpoint + Proxy + AV

- From last year’s Security Ninjutsu Part Two, correlating sysmon with proxy and AV data.
- _raw:

```
[search tag=malware earliest=-20m@m latest=-15m@m | table dest | rename dest as src ]

earliest=-20m@m (sourcetype=sysmon OR sourcetype=carbon_black eventtype=process_launch) OR
(sourcetype=proxy category=uncategorized)

|  stats count(eval(sourcetype="proxy")) as proxy_events count(eval(sourcetype="carbon_black" OR
sourcetype="sysmon")) as endpoint_events by src

| where proxy_events > 0 AND endpoint_events > 0
```

  – 21 seconds

ES Endpoint + Proxy + AV (2)

- tstats:

```
| tstats prestats=t summariesonly=t count(Malware_Attacks.src) as malwarehits from datamodel=Malware where Malware_Attacks.action=allowed groupby Malware_Attacks.src

| tstats prestats=t append=t summariesonly=t count(web.src) as webhits from datamodel=Web where web.http_user_agent="shockwave flash" groupby web.src

| tstats prestats=t append=t summariesonly=t count(All_Changes.dest) from datamodel=Change_Analysis where sourcetype=carbon_black OR sourcetype=sysmon groupby All_Changes.dest

| rename web.src as src Malware_Attacks.src as src All_Changes.dest as src

| stats count(Malware_Attacks.src) as malwarehits count(web.src) as webhits count(All_Changes.dest) as process_launches by src
```

  - 2 seconds

ES Endpoint + Proxy + AV (3)

- Speed Difference: 10.5x
  - It doesn’t always have to be 10,000x. 10x or even 3x is still a huge reduction in resources.
- Query Length difference: 282 characters longer
  - Multiple namespaces can make things longer, and also maybe more complicated sometimes. Worth it though.

Advanced Topics

Because it’s been straightforward so far, right?

allow_old_summaries and summaries_only

- These two settings are perhaps the most important to tstats.
- summaries_only means that we won't automatically fall back to raw data – this means fast results, and much more of a difference than you would probably expect. If searching 100 days of data, and 15 minutes aren't accelerated, we probably don't care.
- allow_old_summaries is key for two scenarios:
  - You leverage the common information model, which is periodically updated, and you want to be able to search data from an earlier version (very likely)
  - You have multiple apps with different global config sharing settings, and you want to search from an app that didn't *generate* the data model originally.

allow_old_summaries and summaries_only (2)

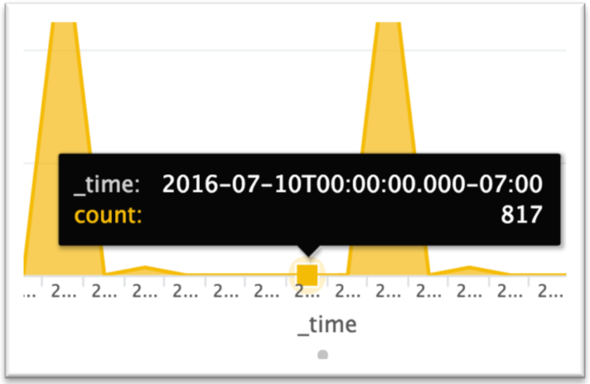
- While these settings are automatically set to true in ES (and probably other Splunk owned apps), because they are so key you may want to set them to true automatically across the system via limits.conf
- Big impact: pivot will use whatever the default is
  - Note: the pivot user interface actually runs tstats. The pivot search command is not impacted – I know, I know.

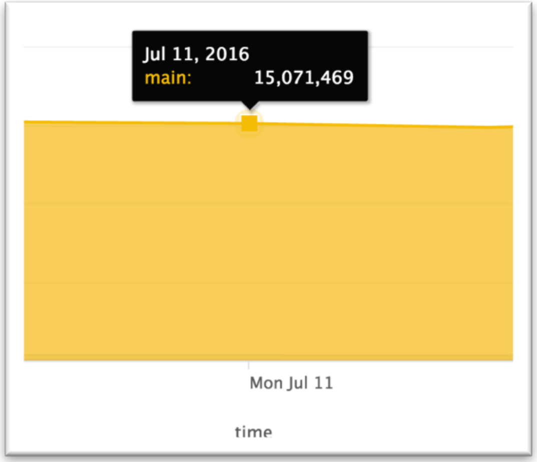
```
[tstats]
summariesonly = <boolean>
* The default value of 'summariesonly' a
* When running tstats on an accelerated
  a mixed mode where we will fall back t
* summariesonly=true overrides this mixe
  TSIDX data, which may be incomplete
* Defaults to false

allow_old_summaries = <boolean>
* The default value of 'allow_old_summar
  command
* When running tstats on an accelerated
  ensures we check that the datamodel se
  is considered up to date with the curr
  that are considered up to date will be
* allow_old_summaries=true overrides thi
  even from bucket summaries that are co
  datamodel.
* Defaults to false
```

prestats=t

- Tstats can be fed into upstream stats. For example, tstats _time span=… put directly into a graph looks terrible.
- | tstats prestats=t count where index=* groupby _time span=1d index| timechart span=1d count by index





chunk_size

- How much data will be retrieved by tstats from a tsidx file at once
- Tradeoff between memory, sorting, and other factors
- Default value (10000000 – 10 MB) is usually the right fit.
  - Lowering that could significantly hurt performance.
  - For very high cardinality, raising it to 50 MB or 100 MB may be beneficial
  - Worth testing out only for a long-running search you will use regularly

Searching Across Multiple Namespaces

- With normal search, you can use as many different indexes, sourcetypes, etc as you want, with reckless abandon.
- With tstats, you can use append=t, but requires prestats=t. Frequently requires munging with eval along the way.
- | tstats prestats=t dc(All_Traffic.dest) from datamodel=Network_Traffic groupby All_Traffic.src
  | tstats prestats=t append=t count from datamodel=Malware groupby Malware_Attacks.dest
  | eval system=coalesce('All_Traffic.src', 'Malware_Attacks.dest')
  | stats dc(All_Traffic.dest), count by system

Searching Across Multiple Namespaces (2)

- If you are querying the same parameters in the first and second query, such as comparing time spans or looking at two counts, use eval with coalecese to define a field

```
|  tstats prestats=t append=t count from datamodel=Malware where
earliest=-24h groupby Malware_Attacks.dest
| eval range="current"
|  tstats prestats=t append=t count from datamodel=Malware where
earliest=-7d latest=-24h groupby Malware_Attacks.dest
| eval range=coalesce(range, "past")
| chart count over Malware_Attacks.dest by range
```

Searching Across Multiple Namespaces (3)

You can also use different fields, such as count(Malware_Attacks.src), count(web.src), and etc.

- | tstats prestats=t summariesonly=t count(Malware_Attacks.src) as malwarehits from datamodel=Malware where Malware_Attacks.action=allowed groupby Malware_Attacks.src
- | tstats prestats=t append=t summariesonly=t count(web.src) as webhits from datamodel=Web where web.http_user_agent="shockwave flash" groupby web.src
- | tstats prestats=t append=t summariesonly=t count(All_Changes.dest) from datamodel=Change_Analysis where sourcetype=carbon_black OR sourcetype=sysmon groupby All_Changes.dest
- | rename web.src as src Malware_Attacks.src as src All_Changes.dest as src
- | stats count(Malware_Attacks.src) as malwarehits count(web.src) as webhits count(All_Changes.dest) as process_launches by src

Pull Malware Data

Pull Web (Proxy) Data

Pull Endpoint Data

Normalize Field Names

Do Count

Drilldown

- Drilldowns from tstats queries don't often work correctly
- Best to put that in a dashboard where you can manually define the drilldown

```
<title>Hosts with Increased # of Malware Attacks</title>
<search>
  <query>| tstats prestats=t append=t count from datamodel=Malware where earliest=-24h groupby Malware_Attacks.dest
         | eval range="current"
         | tstats prestats=t append=t count from datamodel=Malware where earliest=-7d latest=-24h groupby Malware_Attacks.dest
         | eval range=coalesce(range, "past")
         | chart count over Malware_Attacks.dest by range
         | eval past_daily_avg = past/6
         | eval PercIncrease = 100 * round(current / if(past=0, .1, past_daily_avg), 4)
         | sort - PercIncrease
         | search PercIncrease&gt;100</query>
  <earliest>-7d@h</earliest>
  <latest>now</latest>
</search>
<drilldown>
   <link>/app/search/search?q=index=main%20tag=malware%20dest=$row.Malware_Attacks.dest$</link>
</drilldown>
```

_indextime

- While the Splunk UI doesn’t show _indextime normally, you can use it because it is an indexed field. Just | eval _time=_indextime
- You can’t do aggregations on it, but you can filter!
- Both the time range picker *AND* _indextime apply

```
| tstats count min(_time) as min_time max(_time) as max_time where
    [| stats count as search | eval search="_indextime>" . relative_time(now(), "-7d") | table search]
index=* groupby _indextime
| eval lag=_indextime - (min_time + max_time) / 2
| eval _time = _indextime
| timechart avg(lag)
```

A Special Note About Time

_time is special with tstats, for a couple of reasons:

- You can’t do avg(_time) or range(_time)
- You can do min(_time) and max(_time) and of course groupby _time span=10m (or whatever time)

Cardinality

- Data models are phenomenal with split-by cardinality, e.g.:
  - | tstats avg(bytes) from datamodel=Network_Traffic groupby All_Traffic.dest_ip
- Data models are less great with overwhelming field cardinality, when tracking metric data
- Round off irrelevant data points. If you have temperature to 7 decimal places, but 1 decimal place is all that actually matters, just accelerate that.
  - Don't include the unrounded field in your data model, because then the acceleration will store it and you'll use more disk space.

Scheme on What?

- Data Models are a great combination of schema on read and schema on write.
- As with everything in Splunk, you can flexibly define and change your schema, rebuild tsidx, etc.
- But for accelerated data models, you get all the performance of scheme on write… without losing the flexibility to redefine and rebuild as needed.
  - Obviously, for VERY large datamodels, you might not want to wait for a rebuild, but you can affect moving forward

Quirks of Data Model Acceleration

- Second compression. You can’t look at milliseconds or microseconds for _time without hijinks (separate field and separate filtering)
- Requires stats. It’s called tstats for a reason – there’s no tstatsraw in 6.4.
- | datamodel search command was the devil < 6.4 – much better in newest release
- Interrogating fields is a hassle
- TSIDX trades disk space for performance

Summary

Let’s pull it all together, team

Summary

- Getting started: use accelerated pivot on data models
- Getting started w/ tstats: use tstats on normal indexed data
  - counting events
  - looking for indextime lag
- tstats is actually really easy
- That said, there are some weird quirks.
  - Check out the PDF

THANK YOU

.conf2016

splunk>