splunk>

Splunk Scheduled Search Management Guide

Author: David Paper,
Sr. Escalations Manager & Technical Smokejumper
dpaper@splunk.com
Date: 2019-08-07
Version: 1.0.0

Table of Contents

1. ABSTRACT

Splunk is a data analytics tool that makes machine data easily consumable. To achieve this, searches against data can be scheduled, which provides quick response to customer requests, 24/7 alerting, and scheduled report delivery outside of Splunk. The need to manage these scheduled reports grows as Splunk search usage increases. In this discussion, we will cover two broad areas: identification of issues and remediation options. In the identification section, we will review how to measure search skipped and completed searches, determine why a search was skipped, and how to visualize the scheduled search load on the system. In the remediation section, we will cover consolidation of disparate searches where reasonable, review configuration options and redistribution methods to more evenly utilize Splunks search and indexing tiers, and finish with other factors that impact scheduled searches completing successfully.

2. IDENTIFICATION

2.1 PRIORITIZATION

Splunk differentiates between the type of searches run to assist in determining which searches should run when the system begins to run up against config limitations imposed on search concurrency. The four types of searches, and the order Splunk prioritizes them are

1. Ad hoc historical searches. These are searches run manually by users.
2. Manually scheduled reports and alerts with real-time scheduling. These are searches users manually schedule and pick the frequency to run. This is the highest priority of scheduled searches.
3. Manually scheduled reports and alerts with continuous scheduling. These are searches that populate summary indexes and other reports where there can't be gaps in collection of data.
4. Automatically scheduled reports. These are searches that Splunk scheduled and are what power report acceleration and accelerated data models. These searches will fill in any gaps from previous attempted runs the next time they are run.

See http://docs.splunk.com/Documentation/Splunk/latest/Report/Configurethepriorityofscheduledreports#The_priority_order_of_different_types_of_searches for a more detailed discussion on search prioritization.

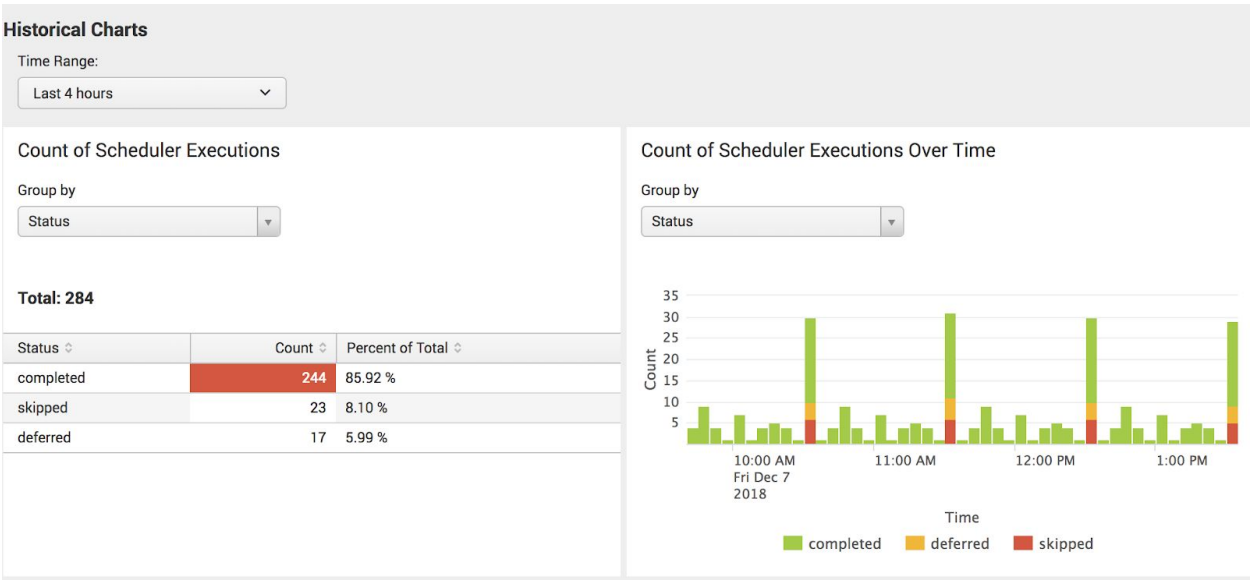
2.2 MEASUREMENT

Peter Drucker hit the nail on the head when he said, "If you can't measure it, you can't improve it." Splunk logs three states for a scheduled search: completed (good), deferred (not bad, just delayed), and skipped (bad). There is also a reason logged for the deferred and skipped states which provide pointers for how to resolve them once the reason is known.

2.2.1 Monitoring Console

The Monitoring Console (MC) in Splunk provides a view ***(Monitoring Console -> Search -> Scheduler Activity: Instance)*** into the situation. Beginning with a view of all three search states. The ideal state is all searches are completed, with no deferrals and no skipped searches.

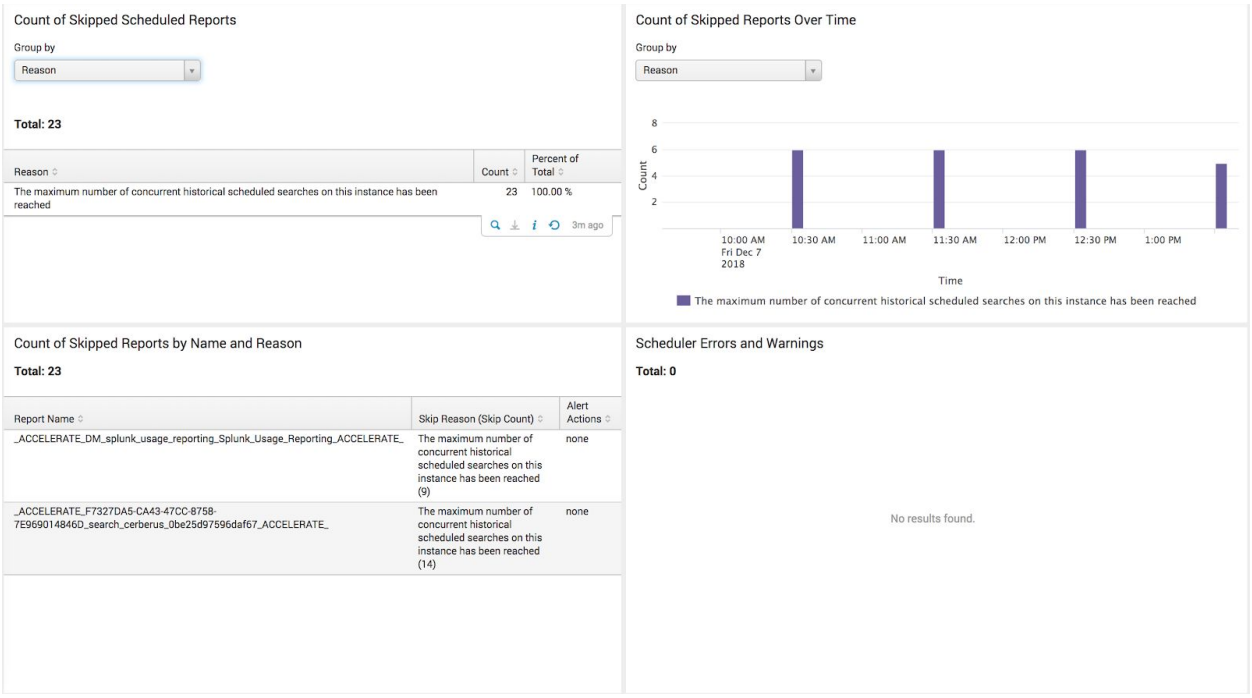
Under Historical Charts, we see a mix of completed, skipped and deferred on this server, with a distinct pattern that emerges for when searches are skipped.



Now we know we have skipped searches. The next logic question is, “Why do we have skipped searches?” There are several reasons to skip a search

1. User disk quota has been reached
2. User role search quota has been reached
3. Maximum searches system wide have been reached (ad hoc + scheduled)
4. Max concurrent historical searches have been reached (searches that are non summarization searches)
5. Max auto summarization searches have been reached (searches that are report accelerations and data model accelerations)

Each search has a reason associated. The breakdown can be found on the same **Scheduler Activity: Instance** page in MC. In the example below, skipped searches show up each hour at 30 minutes past, and end up skipping for the same reason.



2.2.2 Reasons

Search to find log messages related to scheduled search completion, which are surfaces in MC. In the messages will be both a status and the reasons for a search being something other than completed.

```
index=_internal host=search_head sourcetype=scheduler (status="completed" OR status="skipped" OR status="deferred")
```

From scheduler.log

Completed (success)	01-02-2019 18:41:23.130 +0000 INFO SavedSplunker - savedsearch_id="nobody;search;Splunk Messages Saver", search_type="", user="user1", app="search", savedsearch_name="Splunk Messages Saver", priority=default, **status=success**, digest_mode=1, scheduled_time=1546454481, window_time=0, dispatch_time=1546454481, run_time=1.497, result_count=216, alert_actions="", sid="scheduler__user1__search__RMD5852ab06248460647_at_1546454481_72113_B5A72FB9-2872-4F10-B7A3-6AD655122EE9", suppressed=0, thread_id="AlertNotifierWorker-0"
Deferred (continued)	01-02-2019 18:51:43.582 +0000 INFO SavedSplunker - savedsearch_id="nobody;DA-ESS-NetworkProtection;Network - Unusual Volume of Network Activity - Rule", search_type="scheduled", user="admin", app="DA-ESS-NetworkProtection", savedsearch_name="Network - Unusual Volume of Network Activity - Rule", priority=default, **status=continued**, **reason="The maximum number of concurrent running jobs for this historical scheduled search on this instance has been reached"**, concurrency_category="historical_scheduled", concurrency_context="saved-search_instance-wide", concurrency_limit=1, scheduled_time=1546452000, window_time=0
Skipped	01-02-2019 18:52:16.545 +0000 INFO SavedSplunker - savedsearch_id="nobody;search;SecureAuth LDAP Errors", search_type="scheduled", user="user1", app="search", savedsearch_name="SecureAuth LDAP Errors", priority=default, **status=skipped**, **reason="The maximum number of concurrent historical scheduled searches on this instance has been reached"**, concurrency_category="historical_scheduled", concurrency_context="saved-search_instance-wide", concurrency_limit=41, scheduled_time=1546455060, window_time=0

2.2.3 Visualizations

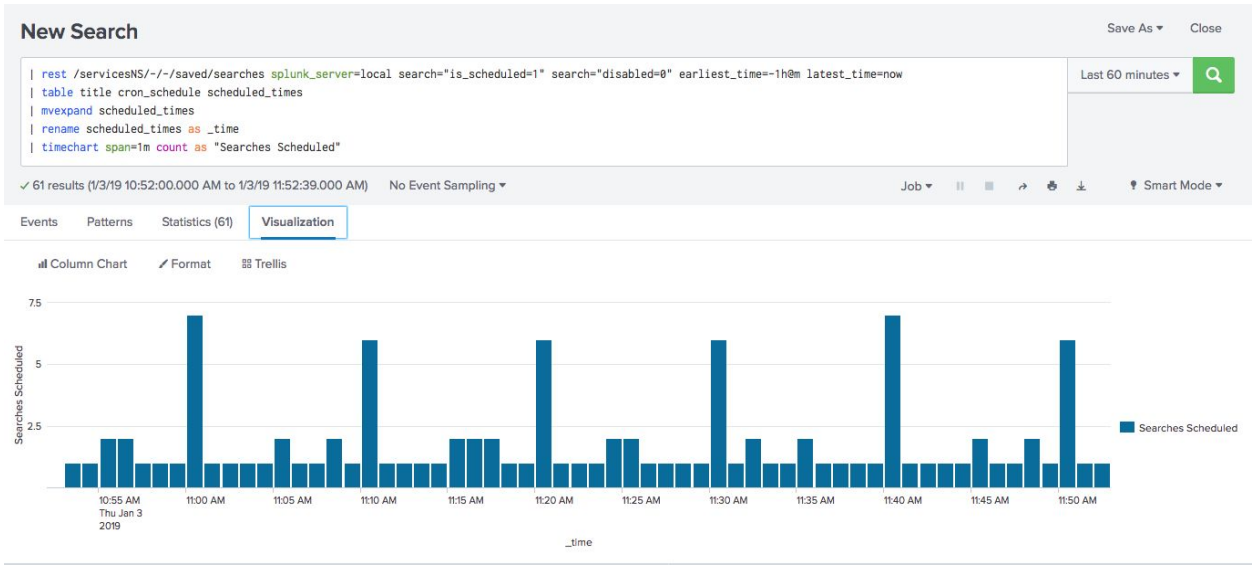
The MC provides several methods to identify skipped search counts, the reasons they skipped, and when they've skipped. Unfortunately, it does not provide tools to the Splunk Admin to adequately visualize what the complete scheduled search picture looks like. As a result, it is difficult to know how bad the problem is until after searches are skipped.

The following searches, run from the search head, create a visual of scheduled searches for the past hour.

To see all searches scheduled at the server wide level

```
| rest /servicesNS/-/-/saved/searches splunk_server=local search="is_scheduled=1"
search="disabled=0" earliest_time=-1h@m latest_time=now
| table title cron_schedule scheduled_times
| mvexpand scheduled_times
| rename scheduled_times as _time
| timechart span=1m count as "Searches Scheduled"
```

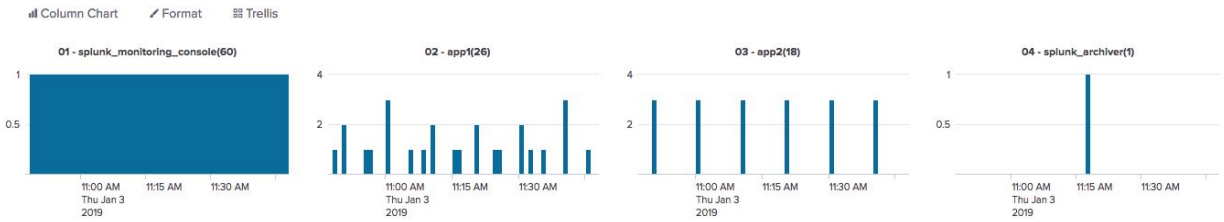
Example results as a bar chart



To see all searches scheduled per app, sorted by total count of searches during the time period

```
| rest /servicesNS/-/-/saved/searches splunk_server=local search="is_scheduled=1"
search="disabled=0" earliest_time=-1h@m latest_time=now
| table title cron_schedule scheduled_times eai:acl.app
| mvexpand scheduled_times
| rename scheduled_times as _time
| rename eai:acl.app AS app
| eventstats count AS total_events by app
| sort - total_events
| streamstats current=f window=1 last(total_events) as prev_eventcount
| fillnull value=0 total_events
| eval tempRank=if(total_events=prev_eventcount,0,1)
| streamstats sum(tempRank) as Rank
| eval Rank=printf("%02d",Rank)
| eval app_name=Rank+" - "+app+"("+total_events+")"
| timechart span=1m count as "Searches Scheduled" by app_name useother=f limit=100
```

Example results as a bar chart and trellis



To see all searches scheduled per user, sorted by total count of searches during the time period

```
| rest /servicesNS/-/-/saved/searches splunk_server=local search="is_scheduled=1"
search="disabled=0" earliest_time=-1h@m latest_time=now
| table title cron_schedule scheduled_times eai:acl.owner
| mvexpand scheduled_times
| rename scheduled_times as _time
| rename eai:acl.owner AS owner
| eventstats count AS total_events by owner
| sort - total_events
| streamstats current=f window=1 last(total_events) as prev_eventcount
| fillnull value=0 total_events
| eval tempRank=if(total_events=prev_eventcount,0,1)
| streamstats sum(tempRank) as Rank
| eval Rank=printf("%02d",Rank)
| eval owner_name=Rank+" - "+owner+"("+total_events+")"
| timechart span=1m count as "Searches Scheduled" by owner_name useother=f limit=100
```

Use these views to identify the apps and individual users that are scheduling searches in a way that is not evenly balanced across the hour. As searches are adjusted via methods discussed in subsequent sections, these views will help validate more event distribution of scheduled search load based on defined configurations.

2.3 BLAME

The two classes of reasons for searches skipping are disk quotas being reached and various concurrent limits being reached.

2.3.1 Quotas

Each individual user has an amount of disk that is available to them to store search results (artifacts). These are managed with user roles. If a user is part of multiple roles, the largest disk quota of all roles they are a member of is used. Disk quotas for each role are not additive.

From scheduler.log

Max quota reached	`01-02-2019 21:28:20.747 +0000 INFO  SavedSplunker - savedsearch_id="nobody;search;READ Resp Time", search_type="scheduled", user="user1", app="search", savedsearch_name="READ Resp Time", priority=default,`

	status=skipped, reason="Search not executed: **The maximum disk usage quota for this user has been reached.** Use the Job Manager to delete some of your saved search results.", **usage=307MB**, quota=100MB, user=user1, concurrency_category="historical", concurrency_context="user_instance-wide", scheduled_time=1546464420, window_time=0

Large searches that return a lot of results (including using verbose mode) will use up disk quota faster. Options to remedy this include

- Raising the roles disk quota
- Refactor one or more of the users scheduled searches so they use less space
- Disable one or more of the scheduled searches for that user
- Adjust the dispatch.ttl setting for one or more of the scheduled searches for that user to lower the amount of time artifacts are held in dispatch (***Settings -> Searches, Reports, and Alerts -> Find the search and open it with Advanced Edit -> dispatch.ttl***)

2.3.2 Roles

User role based search concurrency limits are often hit due to users scheduling many searches at the same time. A frequency behavior is a user setting up several scheduled searches and selecting “Every 5 minutes” in the simple scheduler. Regardless of how many are scheduled, Splunk will adhere to the maximum concurrency limit for that user and the roles they belong to. User role search concurrency behaves like quotas when a user is part of multiple roles. Highest search quota for any role the user belongs to is what is used.

From scheduler.log

Max concurrent historical search	01-02-2019 22:17:38.647 +0000 WARN DispatchManager - Queued job id = user1__user1_dmdfZ2lmc19wcm9k__search13_1546467458.3131852, search = 'search index=_internal | stats count' , **reason = "The maximum number of concurrent historical searches for this user based on their role quota has been reached. concurrency_limit=8"**, provenance = UI:Dashboard:my_internal_stats

Raising the search quota for the user is one obvious solution, but may not be the right one as it may have consequences for the entire search head. The higher an individual users search concurrency, the quicker all search capacity on that search head will be consumed.

2.4 SYSTEM WIDE LIMITS

The final types of scheduled search concurrency limits are search head wide settings.

A system wide of max search concurrency limit a user may hit when scheduled searches attempt to run is "Maximum system wide searches has been reached." This occurs when there are enough users running searches (ad hoc, scheduled, summary) that all available search slots on the search head are full.

From scheduler.log

Max concurrent system wide	02-22-2019 15:40:29.584 +0000 WARN DispatchManager - Queued job id = uqma__uqma_dmdfY3RvX25nYV9wcm9k__search3_1550850028.8090896, search = 'search index=aws_ops_prod sourcetype="aws:PcfAppStats" app_name="/offer-adoption.webservice*" | dedup build_metadata | sort build_metadata' , **reason = "The maximum number of concurrent historical searches**

	on this instance has been reached. concurrency_limit=78", provenance = UI:Dashboard:aws_nga_pinpoint

Alleviating this type of error is more involved. If the Splunk admin wants to raise the overall limit, they will need to evaluate whether the SH has adequate CPU, memory and disk resources to handle more concurrent searches (MC can help here). If it does, the admin will also need to determine if the indexing tier has available headroom to handle more concurrent searches. Indexing tier headroom here comprises of both available CPU capacity (50%, 75%, 90% busy during the busiest minute each day?) and IOPS capacity (if IOPS is already hitting a max plateau, additional search traffic will only slow everything down).

To raise the system wide concurrency will require tweaking limits.conf entries for **base_max_searches** and **max_searches_per_cpu** on the search head. With a large number of cores, setting **max_searches_per_cpu** to "0", and using **base_max_searches** to set exactly how many searches to run will allow fine grain control vs having to work with a multiple of the cores visible to the OS.

Example limits.conf for a 24 core SH:

[search]
max_searches_per_cpu = 0
base_max_searches = 30

Splunk has a couple of built in settings to attempt to ensure that scheduled searches (both summarizing and non-summarizing) don't take up too much capacity from ad hoc searches. As a result, there are a two types of limits that a user may encounter when running a scheduled search that can be updated to allow the search head to better utilize unused capacity for scheduled searches.

A reason of "`Max concurrent historical searches have been reached`" is an indicator that Splunk has exhausted all available search slots that are set aside for non-summarization scheduled searches. By default, this is set to 50%, so if total searches Splunk can run is 22, then only 11 can be non-summarized scheduled searches at any time. A reason of "`Max auto summarization searches have been reached`" is an indicator that all of the available search slots for summarization searches are being used. By default, this is also set to 50%.

From scheduler.log

System wide maximum scheduled concurrent log message	01-02-2019 23:11:04.635 +0000 INFO SavedSplunker - savedsearch_id="nobody;SplunkforPaloAltoNetworks;_ACCELERATE_DM_SplunkforPaloAltoNetworks_pan_endpoint_ACCELERATE_", search_type="datamodel_acceleration", user="nobody", app="SplunkforPaloAltoNetworks", savedsearch_name="_ACCELERATE_DM_SplunkforPaloAltoNetworks_pan_endpoint_ACCELERATE_", priority=default, status=skipped, reason="**The maximum number of concurrent historical scheduled searches on this instance has been reached**", concurrency_category="historical_scheduled", concurrency_context="saved-search_instance-wide", **concurrency_limit=21**, scheduled_time=1546470600, window_time=0
Max auto-summarization skipped search log message	01-02-2019 23:08:47.560 +0000 INFO SavedSplunker - savedsearch_id="nobody;SplunkforPaloAltoNetworks;_ACCELERATE_DM_SplunkforPaloAltoNetworks_pan_firewall_ACCELERATE_", search_type="datamodel_acceleration", user="nobody", app="SplunkforPaloAltoNetworks", savedsearch_name="_ACCELERATE_DM_SplunkforPaloAltoNetworks_pan_firewall_ACCELERATE_", priority=default, status=skipped, reason="**The maximum number of concurrent auto-summarization searches on this instance has been reached**", concurrency_category="summarization_scheduled", concurrency_context="saved-search_instance-wide", **concurrency_limit=10**, scheduled_time=1546470300, window_time=0

3. REMEDIATION

3.1 CONFIGURATIONS

3.1.1 limits.conf

Looking at the system wide view, one of the easiest ways to alleviate skipped searches is to allow Splunks scheduler greater flexibility when there are fewer higher priority searches running. By default, Splunk reserves half of the available search slot capacity for ad-hoc searches, so as to minimize the possibility of a user having their searches queued due to too many scheduled searches already running. There are two settings that will allow Splunk to utilize more search slots for scheduled searches when ad-hoc searches aren't taking them up if the customer workload is more heavily skewed towards scheduled searches than ad-hoc searches.

3.1.1.1 Premium app SH considerations

Not all SHs in an environment should be changed. Enterprise Security and ITSI, which heavily rely on scheduled and accelerated searches to function, can benefit from the changes. An ad-hoc only SH may benefit if scheduled searches are frequently skipped and if the reason is "`Max concurrent historical searches have been reached`" or "`Max auto summarization searches have been reached`". In limits.conf, **max_searches_perc** and **auto_summary_perc** are default set to 50 (unit is percent, but not necessary to include a % in the setting).

Example limits.conf settings for an ES SH:

[scheduler]
max_searches_perc = 75
auto_summary_perc = 100

These settings allow 75% of search slots to be used by scheduled searches, and 100% of those scheduled searches could be auto summarization searches.

3.1.2 savedsearches.conf

Looking at the search level view provides us with three additional options for helping the Splunk scheduler.

schedule_window (Splunk 6.3+) for all searches, so Splunk can reschedule a search for a later time when there are search slots available. Using **schedule_window = auto** allows Splunk's scheduler the maximum flexibility to move searches into the future. Splunk adjusts the earliest & latest time to compensate for any scheduling adjustment so the original times are retained. This works well for infrequently scheduled searches (like every 6 hours or once a day) that need to run, but are less time sensitive for their completion. This kicks in only when the search head has run out of available search slots. Think of this as a macro (high level) configuration as Splunk could potentially move searches around by many minutes or hours.

There is a need to use non-default roles to be able to fully utilize the **schedule_window = auto** setting. All roles that inherit the default user role will have **edit_search_schedule_window = enabled** which will nullify this setting. See savedsearches.conf.spec in the schedule_window section.

allow_skew (Splunk 6.6+) for all non-acceleration searches, so Splunk can adjust the scheduling for frequently scheduled searches. This applies to the 2nd & 3rd priority scheduled search types noted earlier. Using **allow_skew = 5m** allows Splunk to evaluate all frequently scheduled non-acceleration searches, and push some number of them into the future. Splunk adjusts the earliest & latest time to compensate for any scheduling adjustment so the original times are retained. **allow_skew** works at a one second granularity for scheduling slots, allowing all 60 seconds of a minute to be utilized from a scheduling perspective. Searches that had previously been attempted to start at the top of the minute to now become evenly distributed throughout all 60 seconds of that minute. Think of this as a micro (low level) configuration as Splunk will better utilize all 60 seconds each minute to kick searches off. Note that if a search is scheduled more frequently than the **allow_skew** setting, Splunk automatically truncates the **allow_skew** to match the frequency of the scheduled search — **allow_skew = 5m** and a search scheduled to run every 2 minutes would result in applying **allow_skew** for that search with a maximum of 2 minutes instead of 5.

Additionally, **allow_skew** can take a percentage instead of a fixed time value. Be aware that an **allow_skew = 5%** settings will a skew of 1.5 days for a search scheduled every month. An hourly search will result in a 3 minute skew.

Example savedsearches.conf:

[default]
schedule_window = auto
allow_skew = 5m

Additional discussion about **schedule_window** and **allow_skew** can be found at

- https://docs.splunk.com/Documentation/Splunk/latest/Report/Skewscheduledreportstarttimes
- https://www.splunk.com/blog/2017/10/10/schedule-windows-vs-skewing.html
- https://conf.splunk.com/files/2017/slides/making-the-most-of-the-splunk-scheduler.pdf

3.1.3 datamodels.conf

acceleration.allow_skew (Splunk 6.6+) acts just like **allow_skew**, except that it applies to Data Model Acceleration searches. This applies to the 4th priority type of scheduled searches noted earlier. The same rules apply for **acceleration.allow_skew** as for **allow_skew**. Since DMAs are scheduled every 5 minutes by default, setting **acceleration.allow_skew = 5m** perfectly matches the default frequency, providing the Splunk search scheduler the maximum amount of flexibility for scheduled searches when search slots are available.

Example datamodels.conf:

[default]
acceleration.allow_skew = 5m

3.2 CONSOLIDATION

One of Splunks selling points is ease of use when writing SPL — you don't have to be an expert to be able to get basic searches to work. Over time, novice users learn to write alerts to look for problems in the environment. A simplified example is a set of searches to look for specific HTTP error codes in an Apache web server log file and send out an email if any HTTP 403, 404 or 500 log messages are found. This would result in 3 saved searches that were all named similarly that all run very similar searches that execute every 5 minutes.

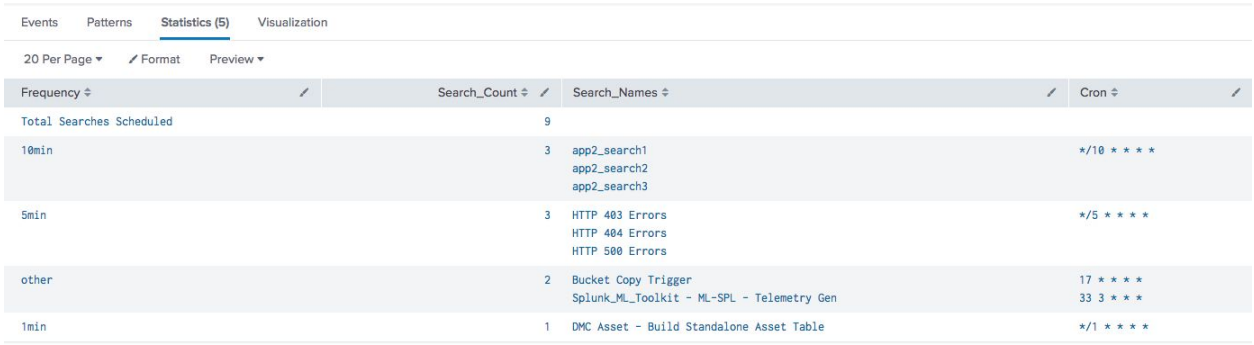
Name	SPL
HTTP 403 Errors	`index=web sourcetype=access_log host=www* status=403 | stats count`
HTTP 404 Errors	`index=web sourcetype=access_log host=www* status=404 | stats count`

HTTP 500 Errors	`index=web sourcetype=access_log host=www* status=500 | stats count`

The difficulty is identifying very similar searches that would be ripe for consolidation without exhaustively reading the SPL of every search. One method is to group them together by the scheduler (“Every 5 minutes”) and look for named searches that are similar.

```
| rest splunk_server=local "/servicesNS/-/-/saved/searches/" search="is_scheduled=1" search="disabled=0"
| fields title, eai:acl.app, eai:acl.owner, cron_schedule, dispatch.earliest_time, dispatch.latest_time, schedule_window, actions
| rename title as "Report_Name", cron_schedule as "Cron_Schedule"
| eval Frequency=if(like(Cron_Schedule,"*/1 %"),"1min",if(like(Cron_Schedule,"* * * * *"),"1min",if(like(Cron_Schedule,"%/5 %"),"5min", if(like(Cron_Schedule,"%/10 %"),"10min",if(like(Cron_Schedule,"*/15 %"),"15min",if(like(Cron_Schedule,"0 %"),"Top of the Hour","other"))))))
| stats count(Report_Name) AS Search_Count values(Report_Name) AS Search_Names values(Cron_Schedule) AS Cron by Frequency
| addcoltotals labelfield=Frequency label="Total Searches Scheduled"
| sort - Search_Count
| table Frequency Search_Count Search_Names Cron
```

Example output from the search above before consolidation makes it easier to find similarly named searches with the same cron scheduling.



Once identified, a slightly more advanced approach would take the 3 similar searches and rewrite them as

```
index=web sourcetype=access_log host=www* (status=403 OR status=404 OR status=500) | stats count by status
```

And then alert on that single search once every 5 minutes, and retire the 3 singular searches.

3.3 REDISTRIBUTION

There are additional methods to take a more manual approach to managing search load. Redistribution of existing scheduled searches and scheduled search collection under a single user and role are outlined below.

3.3.1 Manual Redistribution

If, as a Splunk admin, the idea of using **allow_skew** and **schedule_window** don’t appeal because they take the fine grain control over when a search will run out of your hands, there is an alternate way. Using the example of having 100 searches scheduled at */5 (every 5 minutes), you may want to reasonable redistribute them so that 20 searches run each minute. Using cron pseudo-syntax to create the new settings looks like

0-59/5 * * * * job01 - job20 # Execute at 0, 5, 10, 15, 20 … minutes past the hour
1-59/5 * * * * job21 - job40 # Execute at 1, 6, 11, 16, 21 … minutes past the hour
2-59/5 * * * * job41 - job60 # Execute at 2, 7, 12, 17, 22 … minutes past the hour
3-59/5 * * * * job61 - job80 # Execute at 3, 8, 13, 18, 23 … minutes past the hour
4-59/5 * * * * job81 - job100 # Execute at 4, 9, 14, 19, 24 … minutes past the hour

Evenly balanced. Beautiful. However, manually setting the job schedules for all 100 jobs is tedious, and subject to error. Nico Van der Walt has come up with a SPL based solution to make this process easier.

[Check https://confluence.splunk.com/display/~nvanderwalt/Splunk+Concurrency+tuning+guide to make sure the entry below is still current!]

```
|rest splunk_server=local /servicesNS/-/-/saved/searches
|search is_scheduled=1 disabled=0`
|fields search title author cron_schedule eai:acl.app eai:acl.sharing dispatch.earliest_time
dispatch.latest_time action.summary_index search id
| search NOT (dispatch.earliest_time=rt* OR dispatch.latest_time=rt* OR action.summary_index=1 OR
search=*timechart* OR search=bin OR search=span OR search=*bucket*)
|eval cron_type=case
  (match(cron_schedule,"\*/5 \* \* \* \*"),5,
  match(cron_schedule,"\*/10 \* \* \* \*"),10,
  match(cron_schedule,"\*/15 \* \* \* \*"),15,
  match(cron_schedule,"\*/30 \* \* \* \*"),30,
  match(cron_schedule,"^0 \* \* \* \*"),0,0=0,-1)
| where cron_type >= 0
|sort cron_type
| eval reset_count=case(cron_type=5,4,
                        cron_type=10,9,
                        cron_type=15,14,
                        cron_type=30,29,
                        cron_type=0,59)
| streamstats count reset_after="count=reset_count" by cron_type
| eval lower_bound=count
| eval upper_bound=59
| eval new_cron=case(cron_type=0,count .  " * * * *",
                     cron_type!=0,lower_bound . "-" . upper_bound . "/" . cron_type .  " * * * *")
|eval change_script="curl -k -u $SPLUNK_CREDENTIALS -XPOST " + id + " -d " + "cron_schedule=\"" +
new_cron + "\" || true"
|eval rollback_script="curl -k -u $SPLUNK_CREDENTIALS -XPOST " + id + " -d " + "cron_schedule=\"" +
cron_schedule + "\" || true"
|fields  title author cron_schedule new_cron eai:acl.app eai:acl.sharing change_script
rollback_script
```

Running this produces a table that includes curl commands that can be copied into a script and executed against the SH to let REST calls redistribute searches. Each curl command will update the cron schedule for a single search. In addition to doing every 5 minute searches, it will also do */10, */15, */30 and * (hourly) searches. The admin using this search is encouraged to carefully review the resulting curl commands to ensure that any searches specifically not wanting to or needing to be rescheduled are removed from the list.

Prior to executing either curl command, the variable $SPLUNK_CREDENTIALS will need to be set.

3.3.2 Scheduled Search Collection

As of Splunk 6.6, there is now an easy way to change ownership of a knowledge object from one user to another en masse through the GUI. Using this new functionality allows a new administration pattern to emerge. The idea is to create a user that can own the scheduled searches for

an app, with a role created that has sufficient disk quota and concurrent search quota to accommodate all scheduled searches that live within an app.

Using the “foobar" app as an example

1. Create a role, foobar_search
2. For foobar_search, set a higher than default disk quota (default is 100MB, a good start would be 1GB), this will likely need to be tweaked up or down over time based on environment search load for this user
3. For foobar_search, set a higher than normal concurrent search quota (default is 3, a good start would be 6), this will likely need to be tweaked up or down over time based on environment search load for this user
4. For app, create an app user (may be a local Splunk account or a process account in external auth) which will never need a password change and never expire, called foobar_search (same as the role)
5. Set the foobar_search user to be a member of the foobar_search role
6. In Settings under All Configurations, click on the Reassign Knowledge Objects page, and find the scheduled searches for the foobar app, and assign them to the foobar_search user

3.4 OTHER CONSIDERATIONS

The users concurrent search role quota includes all non-RT searches. Scheduled searches + ad hoc all count towards the maximum number of searches a user can run.

The search below will identify both the current system totals and the concurrency counts for a specified user on a search head. Note that a span of 10 seconds in time chart is the smallest granularity possible.

```
index=_internal earliest=-1h group=search_concurrency host=searchhead1 ("system total" OR user=myuserid)
| rex field=_raw mode=sed "s/system total/user=system/g"
| eval user=coalesce(user,"system")
| timechart max(active_hist_searches) by user
```

Replace `myuserid` with the userid being investigated.

4. CONCLUSION

Splunk makes it incredibly easy to schedule a search for any user on the system. It isn’t quite as easy to manage the menagerie of scheduled searches that collect over time. With the information presented in this paper, the reader should have a better understanding of how search load is managed under the covers, role and system limits are applied, scheduling, skewing and remediation are possible to better utilize the resources available to Splunk.

5. APPENDIX

Several of the searches referenced in this whitepaper, along with several others relevant to managing Splunk search load may be found at https://github.com/dpaper-splunk/public/blob/master/dashboards/extended_search_reporting.xml.

Acknowledgements

David Paper wishes to thank Josh Wisely for his feedback on this discussion.