splunk> .conf2017

Making the Most of the Splunk Scheduler

Paul J. Lucas | Principal Software Engineer, Splunk

September 25–28, 2017 | Washington, DC

Forward-Looking Statements

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 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.

splunk> .conf2017

Personal Introduction

Principal Software Engineer

- On the Core Engineering Team.
- Search Scheduler improvements for Splunk Enterprise.
- Splunk Cloud remote storage.
- Deployment Server.
- Using C++ since the “cfront” days at AT&T Bell Labs.
- Transit enthusiast. 😊

splunk> .conf2017

Intended Audience

- **Who is this presentation for?**

  This presentation is for *Splunk Administrators* of any experience level who provision, monitor, or maintain Splunk Enterprise deployments.

  It's especially for those who are currently experiencing capacity issues such as searches that are either taking a long time to run or are being skipped.

Agenda

- **Scheduled Searches:**
  - Introduction
  - How Cron Works
  - Cron vs. Splunk Scheduler
- **Splunk Scheduler Concepts:**
  - limits.conf Settings
  - Deferred vs. Skipped
  - Latency
- **Splunk Scheduler Details:**
  - How the Splunk Scheduler Works
  - Priority Scoring
- **Splunk Scheduler Tools:**
  - Distributed Management Console
- **Takeaways**

Scheduled Searches

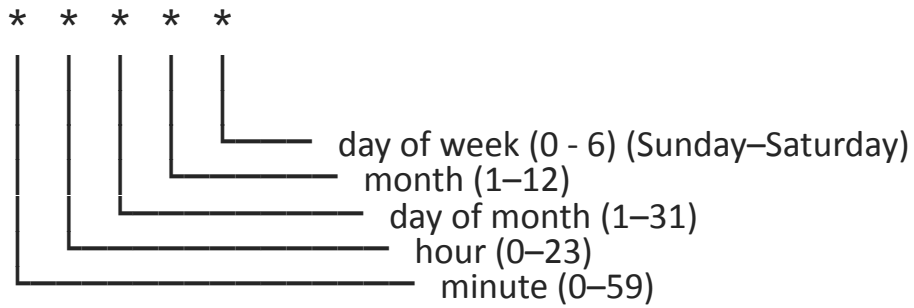
Scheduled Searches: Introduction

- Splunk allows you to save your searches and run them on a schedule.
- Scheduled searches can be used to trigger an alert action (possibly when a condition is met) or to speed-up dashboards.
- An alert action is either sending an e-mail or running a script.
- **Example**: `index=_internal source=*splunkd.log* error`

Title	Too many errors
Trigger condition	Number of Results
Number of results is	Greater than: 5
in	1 minute

Scheduled Searches: Introduction

- Scheduling is specified via a five-field cron string:



- Field values: all (`*`), number (e.g., `0`), ranges (e.g., `1-5`), lists (e.g., `1,8,15,22`), and "every n" (e.g., `*/6`).
- **Example**: `0 */6 1,15 * *` means every 6 hours on the hour on the 1st and 15th of every month.

How Cron Works

- For each cron entry, calculate the next run-time of the command.
- Place all commands in a priority queue by time.
- Enter main loop:
  - Examine the entry at the head of the queue.
  - Calculate the delta between that entry's next run-time and *now*.
  - If delta > 0, sleep for that period of time.
  - Run the entry's command (in the background).
  - Calculate the *next* run-time of the command and place it back on the queue with that new time value.

Cron vs. Splunk Scheduler

Cron

- No job quotas.
- Entirely manual scheduling — have to skew searches by hand:

```
0  0 * * * command-1
15 0 * * * command-2
30 0 * * * command-3
45 0 * * * command-4
```

- Limited to a single machine.

Splunk Scheduler

- Quotas: limit search concurrency — reserves CPU for other tasks.
- Searches over quota are deferred, but implicitly retried repeatedly for the duration of their periods until either run or skipped.
- Can distribute searches across a cluster of machines.

splunk> .conf2017

Splunk Scheduler Concepts

splunk> .conf2017

`limits.conf` Settings

- **`max_searches_per_cpu`**: Maximum number of concurrent searches per CPU (default = 1).
- **`base_max_searches`**: A constant added to max. total searches (default = 6).
- Given those, the total maximum number of concurrent searches allowed is:

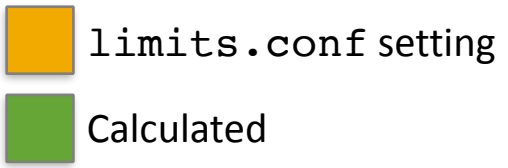
max. total searches = (`max_searches_per_cpu` × *number_of_CPUs* + `base_max_searches`) × *size_of_cluster*

- **`max_searches_perc`**: Maximum number of concurrent searches the scheduler can run as a percentage of max. total searches (default = 50).

limits.conf Settings

- Some example numbers:

CPUs	max_searches_per_cpu	base_max_searches	max. total searches	max_searches_perc	max. scheduled searches
1	1	6	7	50%	3
8	1	6	14	50%	7
64	1	6	70	50%	35



`max_searches_perc` Setting

- **`max_searches_perc`**: Maximum number of concurrent searches the scheduler can run as a percentage of max. total searches (default = 50).
- **Variance (≥6.3)**: Allow `max_searches_perc` to vary by time or day:

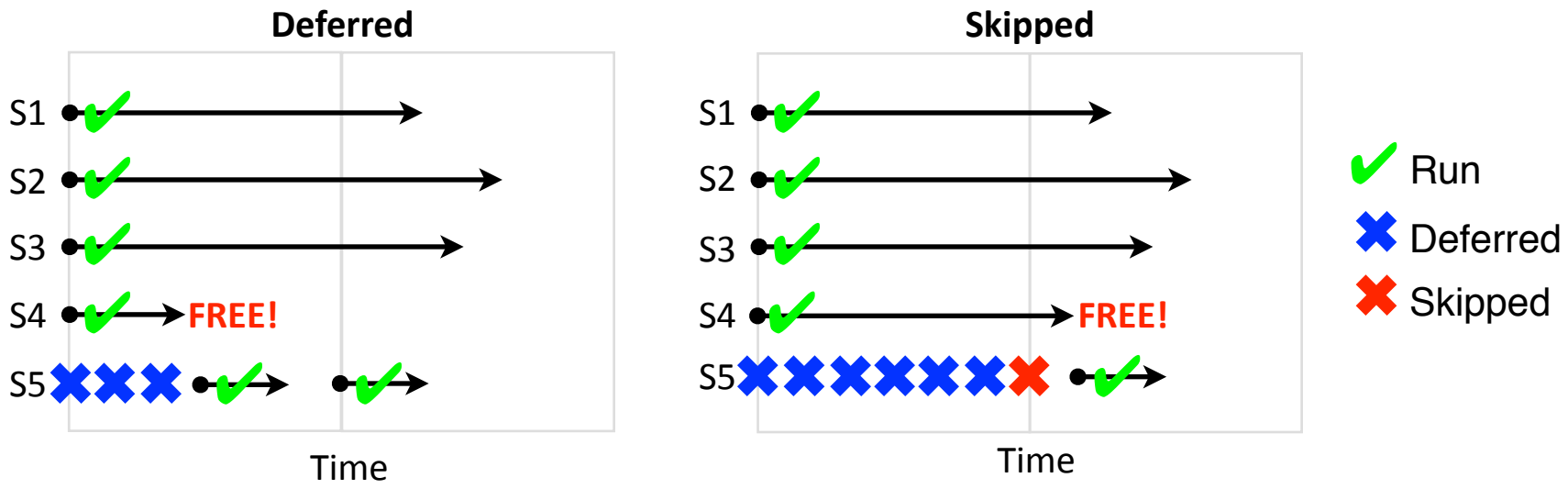
```
max_searches_perc = 50

# Allow value to be 75 anytime on weekends.
max_searches_perc.1 = 75
max_searches_perc.1.when = * * * * 0,6

# Allow value to be 90 between midnight and 5am.
max_searches_perc.2 = 90
max_searches_perc.2.when = * 0-5 * * *
```

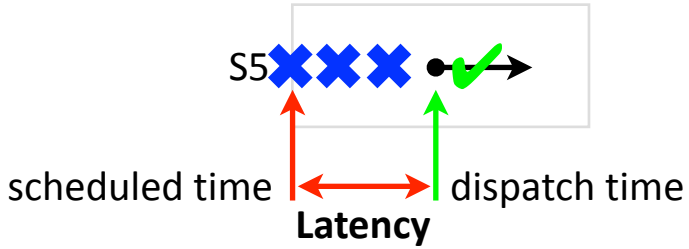
Deferred vs. Skipped

- As mentioned, searches over quota are *deferred*, but are implicitly retried repeatedly for the duration of their periods until either run or skipped.



Latency

- "Latency" is the difference between a search's *scheduled* and *dispatched* times.



- Non-zero latency means scheduler is oversubscribed (at least temporarily).
- Causes delays in alerting and may lead to skipping.
- May be mitigated by *schedule windows* (≥6.3 — more later).

Splunk Scheduler Details

splunk> .conf2017

How the Splunk Scheduler Works

- For each search, calculate the next run-time of the search.
- Place all searches in a `map`<*search_id*,*next_runtime*>.
- Enter main loop:
  - For each search, if its next run-time ≤ *now*, add it to the candidate search list.
  - Randomly shuffle the candidate list.
  - For each candidate search, calculate its *priority score*.
  - Sort all candidate searches by priority score.
  - For each candidate search, if it doesn't exceed quota, run it; calculate the *next* run-time of the search, and update the map.

Priority Scoring

- Multi-term priority scoring (≥6.3) mitigates search latency, skipping, and starvation (when oversubscribed) — improved performance by at least 25%.

$$
\begin{aligned}
score(j) &= next\_runtime(j) \\
&\quad + estimated\_runtime(j) \times \texttt{priority\_runtime\_factor} \\
&\quad - skipped\_count(j) \times period(j) \times \texttt{priority\_skipped\_factor} \\
&\quad + window\_adjustment(j) \\
&\quad - priority\_adjustment(j)
\end{aligned}
$$

Priority Scoring

- Multi-term priority scoring (≥6.3) mitigates search latency, skipping, and starvation (when oversubscribed) — improved performance by at least 25%.

$$
\begin{aligned}
score(j) &= next\_runtime(j) \\
&\quad + estimated\_runtime(j) \times \texttt{priority\_runtime\_factor} \\
&\quad - skipped\_count(j) \times period(j) \times \texttt{priority\_skipped\_factor} \\
&\quad + window\_adjustment(j) \\
&\quad - priority\_adjustment(j)
\end{aligned}
$$

splunk> .conf2017

Priority Scoring

- Multi-term priority scoring (≥6.3) mitigates search latency, skipping, and starvation (when oversubscribed) — improved performance by at least 25%.

$$
\begin{aligned}
score(j) &= next\_runtime(j) \\
&\quad + estimated\_runtime(j) \times \texttt{priority\_runtime\_factor} \\
&\quad - skipped\_count(j) \times period(j) \times \texttt{priority\_skipped\_factor} \\
&\quad + window\_adjustment(j) \\
&\quad - priority\_adjustment(j)
\end{aligned}
$$

Priority Scoring

- Multi-term priority scoring (≥6.3) mitigates search latency, skipping, and starvation (when oversubscribed) — improved performance by at least 25%.

$$
\begin{aligned}
score(j) &= next\_runtime(j) \\
&\quad + estimated\_runtime(j) \times \texttt{priority\_runtime\_factor} \\
&\quad - skipped\_count(j) \times period(j) \times \texttt{priority\_skipped\_factor} \\
&\quad + window\_adjustment(j) \\
&\quad - priority\_adjustment(j)
\end{aligned}
$$

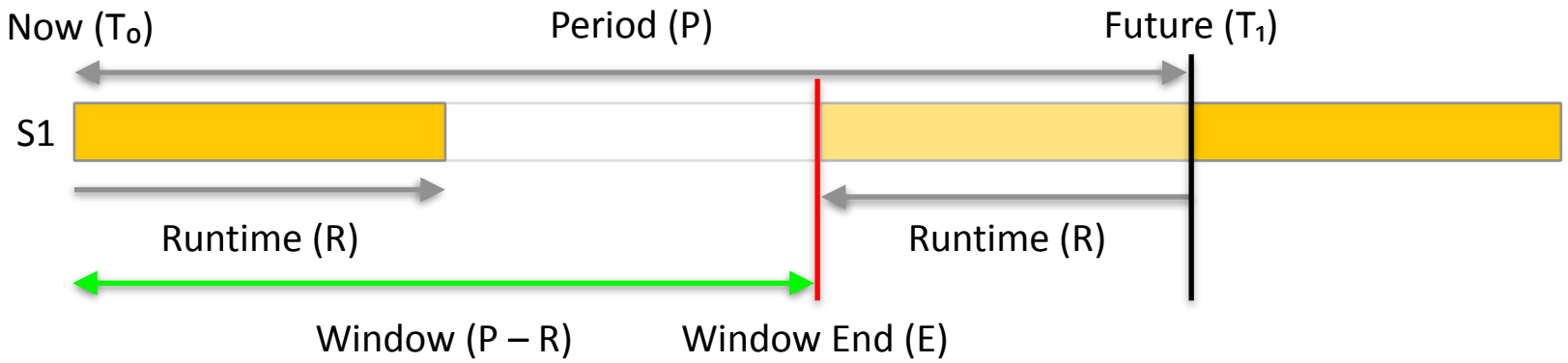
Scoring: Window Adjustment

- **Problem**: Scheduler can't distinguish between searches that (A) *really should* run at a specific time (just like cron) from those that (B) don't have to. This can cause latency or skipping.
- **Solution** (≥6.3): Give a *schedule window* (manually, in minutes) to searches that don't have to run at precise times.

  Example: For a given search, it's OK if it starts running sometime between midnight and 6am, but you don't really care when specifically.

Scoring: Window Adjustment

- **Auto Windows (≥6.5)**: An auto value calculates the maximum window for you.



- S1 can start any time between T0 and E and still finish before its next run at T1.

Scoring: Window Adjustment

Schedule Window key points:

- A search with a schedule window helps *other* searches.
- It's best to use *auto* windows.
- Manual windows require the `edit_search_schedule_window` capability.
- Manual windows *should not* be used for searches that run every minute.
- Manual windows *must* be less than a search's period.
- Priority adjustments (higher, highest) take precedence over windows.
- Windows are *not* a deadline.

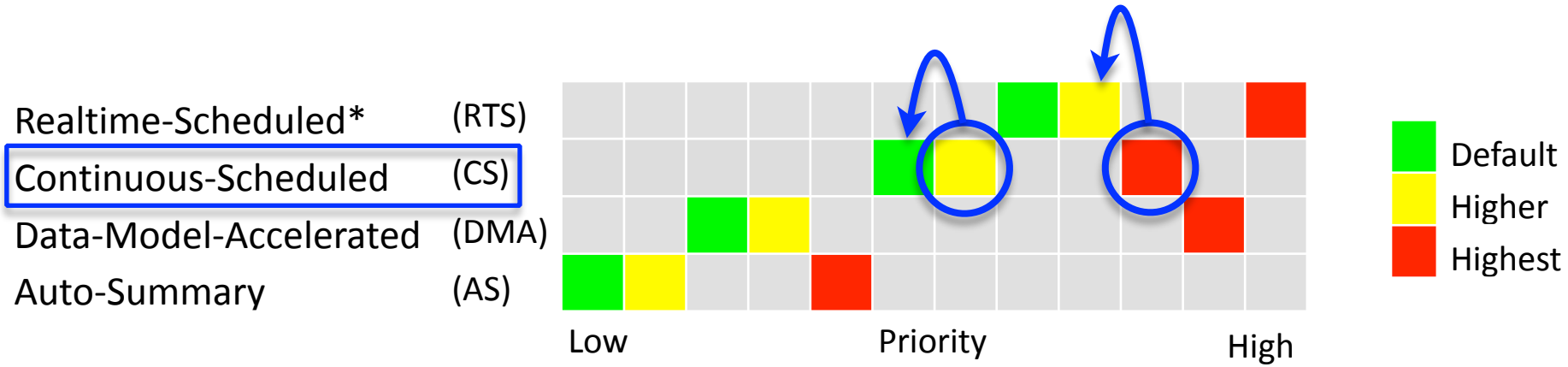
Priority Scoring

- Multi-term priority scoring (≥6.3) mitigates search latency, skipping, and starvation (when oversubscribed) — improved performance by at least 25%.

$$
\begin{aligned}
score(j) &= next\_runtime(j) \\
&\quad + estimated\_runtime(j) \times \texttt{priority\_runtime\_factor} \\
&\quad - skipped\_count(j) \times period(j) \times \texttt{priority\_skipped\_factor} \\
&\quad + window\_adjustment(j) \\
&\quad - priority\_adjustment(j)
\end{aligned}
$$

Scoring: Priority Adjustment

- Scheduled saved searches are stratified into priority tiers:
  Default = same as other default searches as he same tier
  Higher = higher than default searches of the same tier
  Highest = higher than some searches of other tiers



* Most common tier.

Dispatch Time Skewing

- **Problem**: Scheduler dispatches all your searches as soon as possible after the zeroth second of a minute. (For most customers, this is a good thing!) However, for lots of searches that run frequently, this can cause network or other infrastructure saturation.
- **Solution (≥6.6)**: "randomly" skew (large numbers of) your searches so they don't start at the zeroth second. New property in `savedsearches.conf`:

  `allow_skew`

  — A maximum duration *N* (seconds, minutes, hours, days); **OR**:

  — A maximum percentage of period 0–100%.

  Examples:

  ```
  allow_skew = 60s
  allow_skew = 50%
  allow_skew = 100
  ```

  ERROR: no duration unit or `%`

splunk> .conf2017

Dispatch Time Skewing (cont.)

- **Very Skew-able** searches are those that may be skewed by as much as their entire period; they are only those having a `cron_schedule` in one of the following forms:

Min	Hour	Day	Mon	DoW	Meaning
*	*	*	*	*	Every minute
*/N	*	*	*	*	Every N minutes
0	*	*	*	*	Every hour
0	*/N	*	*	*	Every N hours
0	0	*	*	*	Daily (at midnight)

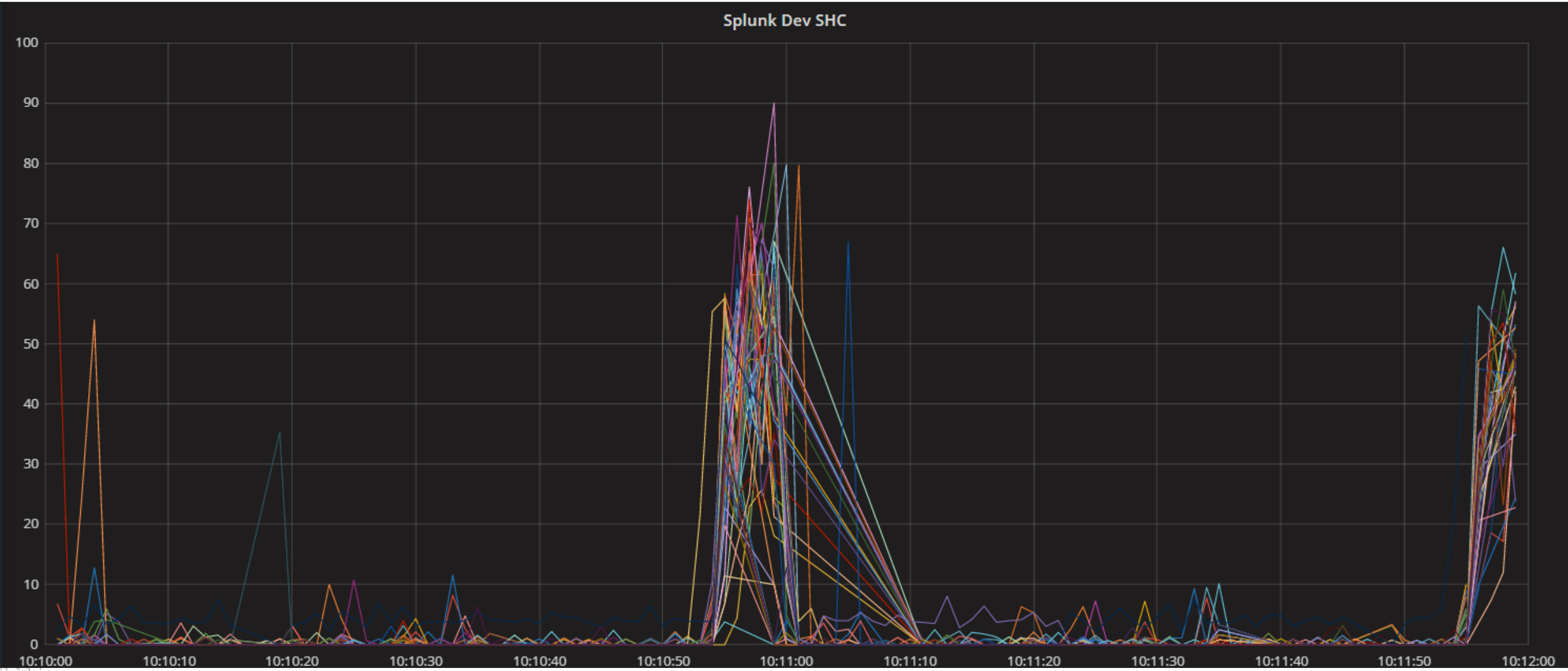
For such searches, it's likely that the user doesn't care at what *actual* minute or hour the search runs just so long as it's *once per* N minutes/hours.

splunk> .conf2017

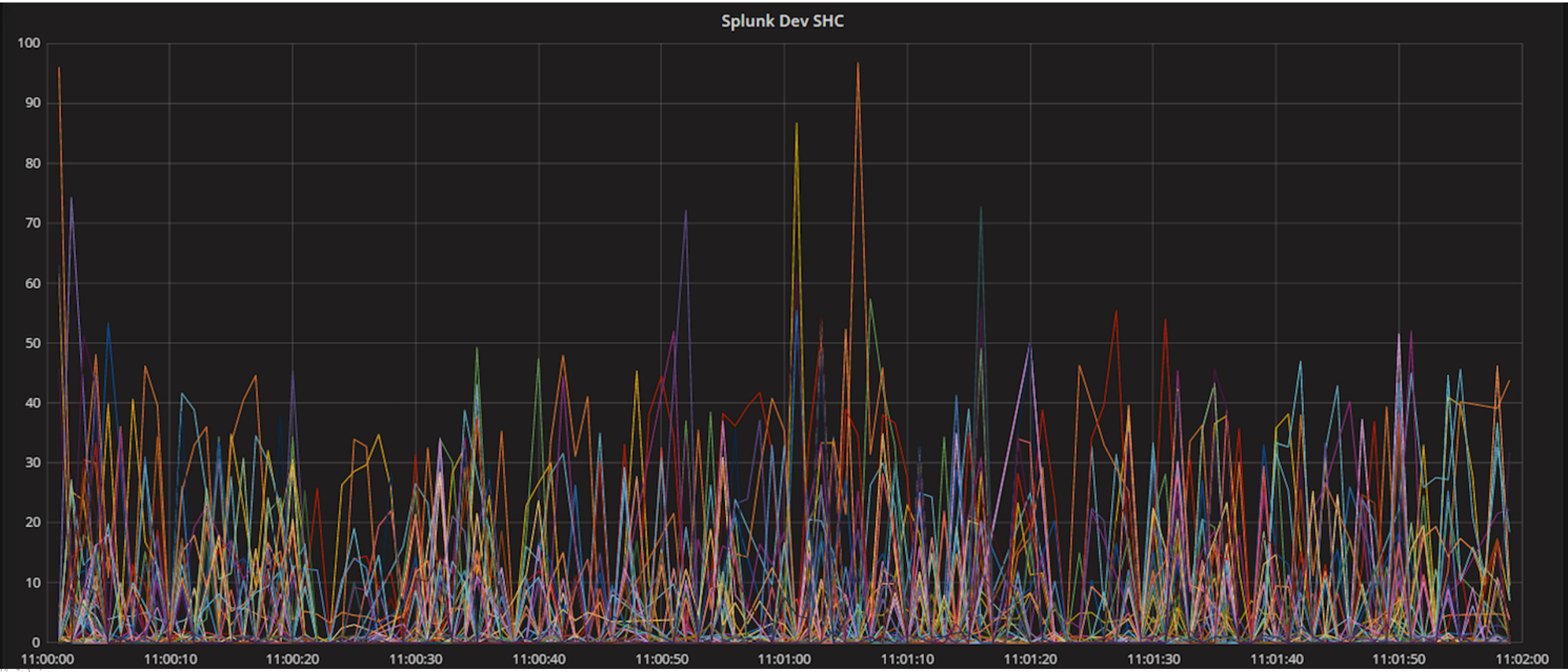
Dispatch Time Skewing (cont.)

- **Somewhat skew-able** searches are those that do *not* have `cron_schedule` strings among one of the aforementioned forms meaning they may be skewed by at most 60 seconds.
- **Rationale**: `cron_schedule` strings that contain either specific (non-zero) minute(s) or hour(s) shall be assumed to have been *precisely* specified by the user to run at those *specific* times. Therefore, the scheduler should not skew such searches (much).

Dispatch Time Skewing (before)



Dispatch Time Skewing (after)



Dispatch Time Skewing (cont.)

- **What about `max_searches_perc`?**
  Before skewing, lowering `max_searches_perc` was believed to be a way to solve this problem. Doing so should cause the searches it can't run now to be run later. However, the problems with using `max_searches_perc` this way are:
  1. A Splunk Administrator would have to use trial-and-error to find a "Goldilocks" value: too high and "spikiness" still happens; too low and searches may get skipped.
  2. `max_searches_perc` is intended to reserve CPU for ad-hoc searches, so using it to prevent "spikiness" isn't what it was intended for.

Dispatch Time Skewing vs. Schedule Windows

- **Skewing solves a slightly different (and rarer) problem than windows (≥6.3).**
  - **Schedule windows** convey searches' *lesser importance* allowing searches of greater importance to have a better chance of running under resource-constraint.
  - **Search Skewing** spreads searches *regardless of importance* out over a period of time so they collectively don't overwhelm hardware.
- Windows and skewing are *independent*.
- Skewed searches are ***still*** subject to windows.

Splunk Scheduler Tools

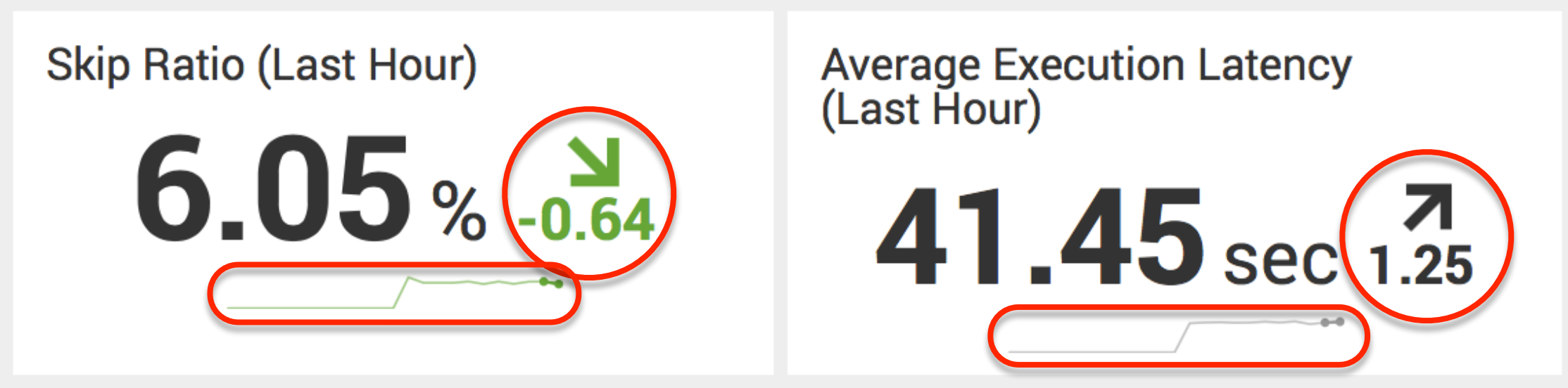
splunk> .conf2017

Distributed Management Console (DMC)

- The *Distributed Management Console* (DMC) is the way to monitor a Splunk Enterprise deployment — including the search scheduler (≥6.4).
- To access the DMC: *Settings (menu)* > *Monitoring Console (icon)* > *Scheduler* > *Scheduler Activity: Instance/Deployment*.
- There are many numbers and charts there — too many to cover here — so I'll just cover the two that I think are the most important:
  1. *Skipped Searches*.
  2. *Latency*.

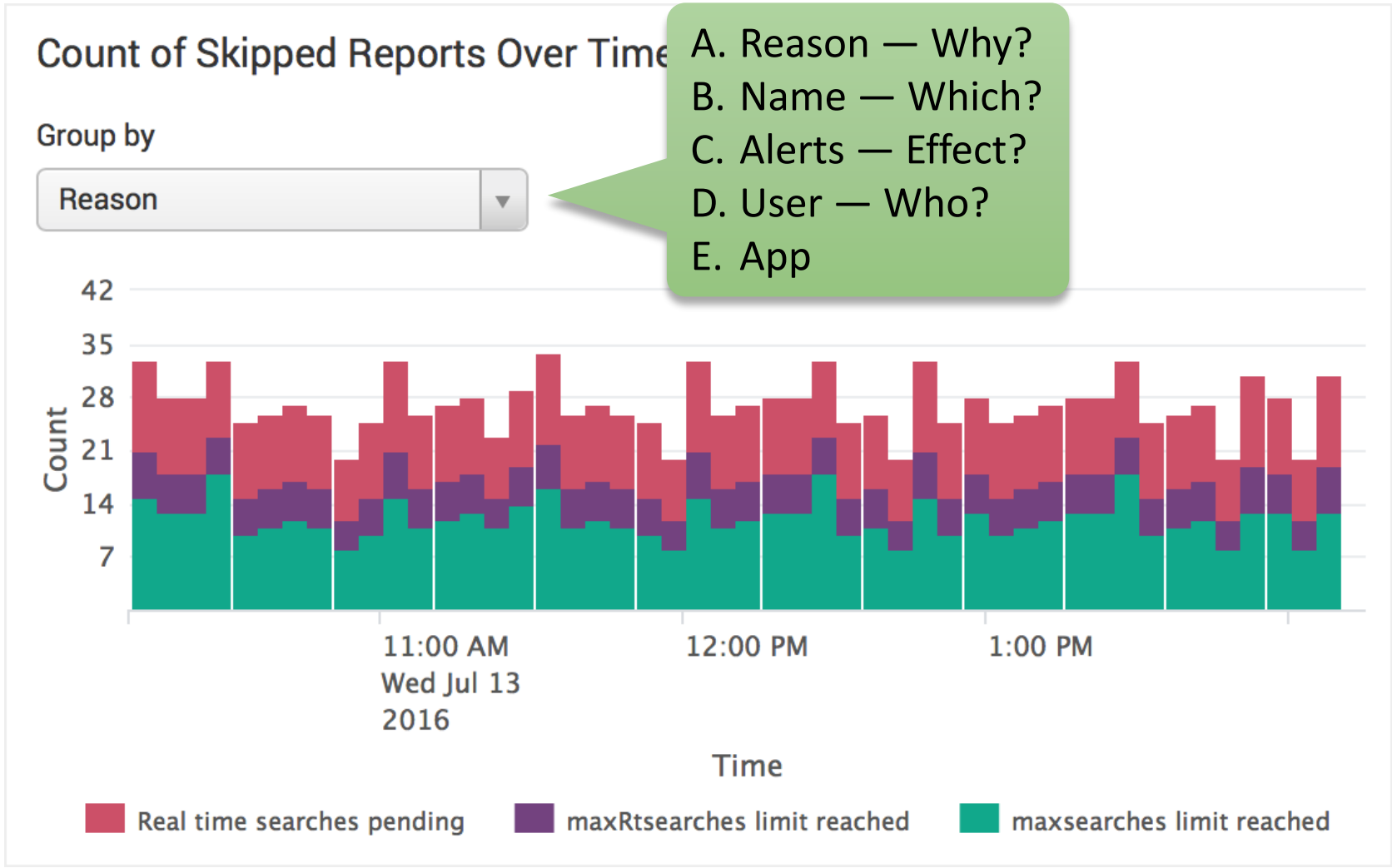
DMC Scheduler Activity

- At the top of the DMC page, there are several numbers. Two of the most important are *Skip Ratio* and *Average Execution Latency*.



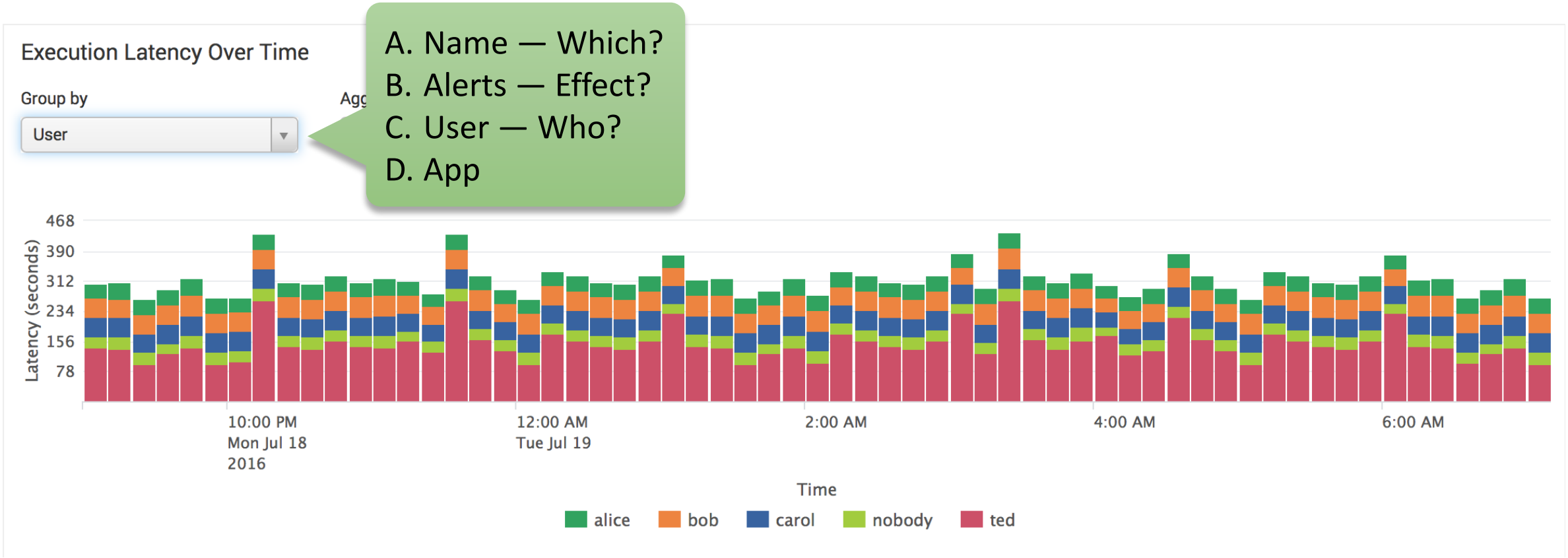
DMC Scheduler Activity: Skipped Searches

- **What this chart shows:** Discretized counts of skipped searches.



DMC Scheduler Activity: Latency

- **What this chart shows:** Discretized amounts of latency.



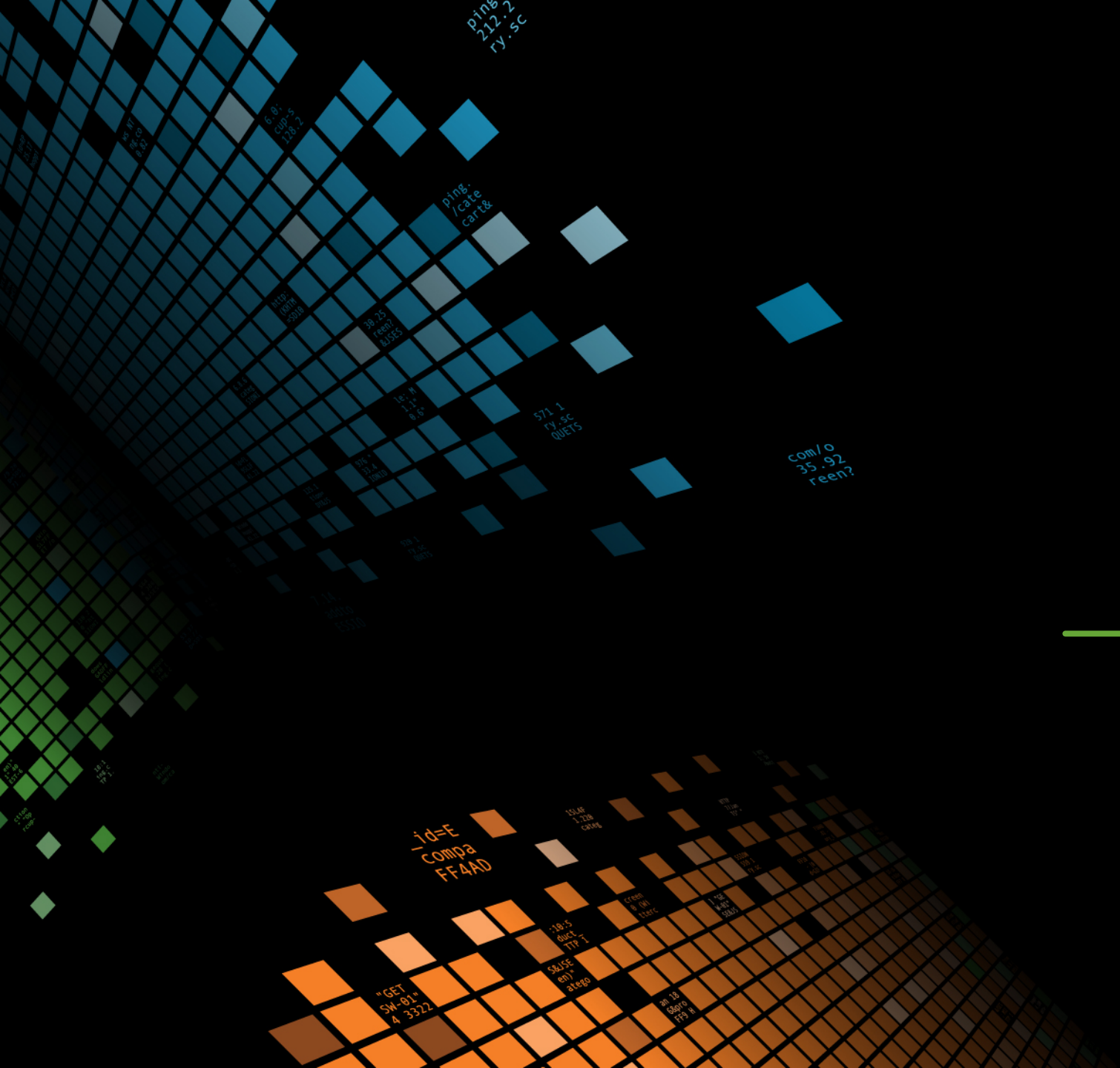
Key Takeaways

1. Recent Splunk Enterprise versions added better priority scoring and search windows for much improved search scheduling by at least 25%.

2. For infrequent searches (hourly, daily, etc.) use schedule windows, preferably auto windows.

3. Use the DMC (under *Settings (menu)* > *Monitoring Console (icon)* > *Scheduler* > *Scheduler Activity: Instance/Deployment*) to monitor scheduler performance: lots of skipped searches or high latency is bad.

4. If, despite tuning, you still have frequently skipped searches or high latency, then you probably need a bigger CPU or more machines in your cluster.



Q&A

splunk> .conf2017

Thank You

Don't forget to rate this session in the .conf2017 mobile app

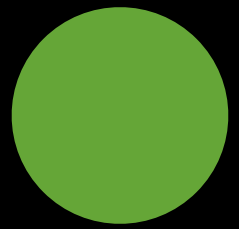
splunk> .conf2017

Line and Shape Assets

Copy/paste these graphics to use in your own presentations

Dark background assets

Dark background overlay



Icon placeholder

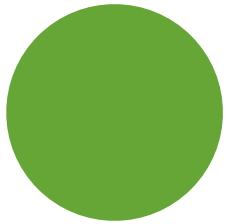
Green Line, 1pt, Cap type: Round

Gray 25% Line, 1pt, Cap type: Round

Gray 25% Line, 1pt, Cap type: Round

White background assets

White background overlay,
Gray 80%, Accent 3, Transparency 85%



Icon placeholder

Green Line, 1pt, Cap type: Round

Gray 25% Line, 1pt, Cap type: Round

Gray 25% Line, 1pt, Cap type: Round

splunk> .conf2017