What’s New In The Splunk Scheduler

Paul J. Lucas

Principal Software Engineer, 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.

Personal Introduction

Paul J. Lucas
Principal Software Engineer, Splunk

- On the Core Server Engineering Team.
- Does Search Scheduler improvements for Splunk Enterprise.
- Does remote storage for Splunk Cloud.
- Did parts of the Deployment Server.
- Has been using C++ since the "cfront" days at AT&T Bell Labs.
- Is a transit enthusiast. 😊

Agenda

- **Splunk Scheduler Details**:

  Priority Scoring Changes

  - Auto Windows
- Priority Adjustments
- **Splunk Scheduler Tools**:

  Distributed Management Console (DMC)
- **Takeaways**

Splunk Scheduler Details

.conf2016

splunk>

How the Splunk Scheduler Works

1. For each search, calculate the next run-time of the search.
2. Place all searches in a `map`<*search_id*`,`*next_runtime*>.
3. Enter main loop:
    A. For each search, if its next run-time ≤ *now*, add it to the candidate search list.
    B. Randomly shuffle the candidate list.
    C. For each candidate search, calculate its *priority score*.
    D. Sort all candidate searches by priority score.
    E. 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%.

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

Priority Scoring

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

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

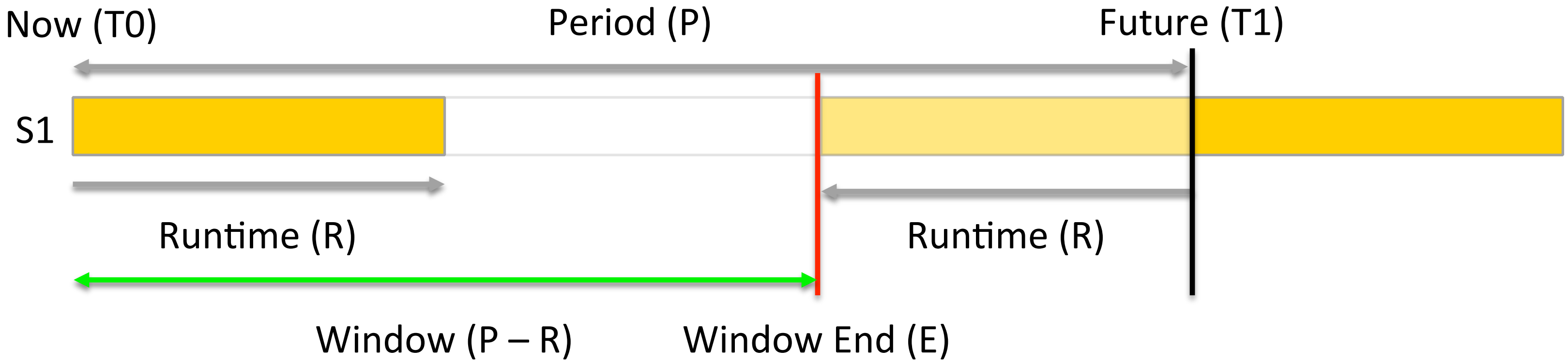
Scoring: Window Adjustment

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

Priority Scoring

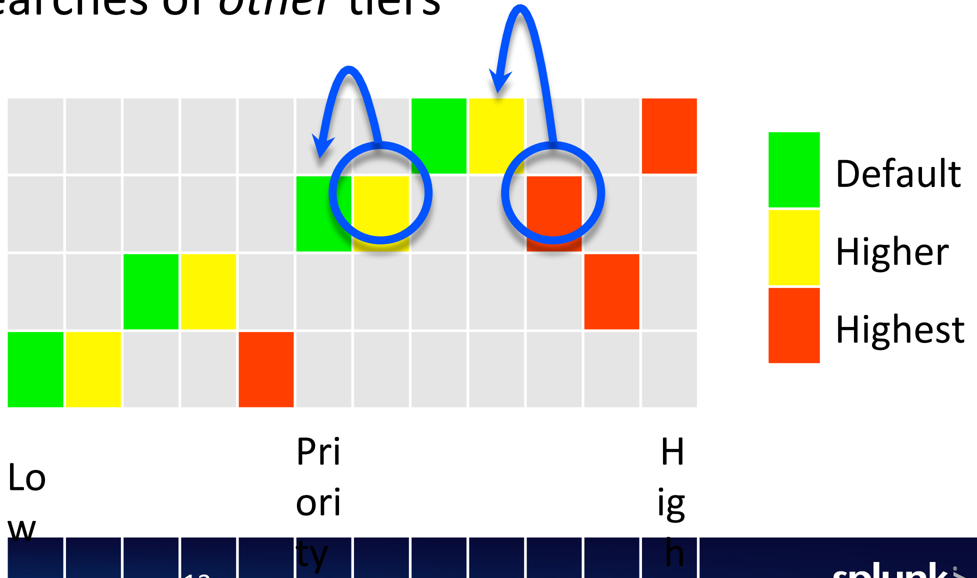
- Multi-term priority scoring (≥6.3) mitigates search latency, skipping, and starvation (when oversubscribed) — improved performance by at least 25%.

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

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

Realtime-Scheduled*	(RTS)								🟩	🟨			🟥
Continuous-Scheduled	(CS)						🟩	🟨			🟥		
Data-Model-Accelerated	(DMA)			🟩	🟨							🟥	
Auto-Summary	(AS)	🟩	🟨			🟥							



* Most common tier

Splunk Scheduler Tools

.conf2016

splunk>

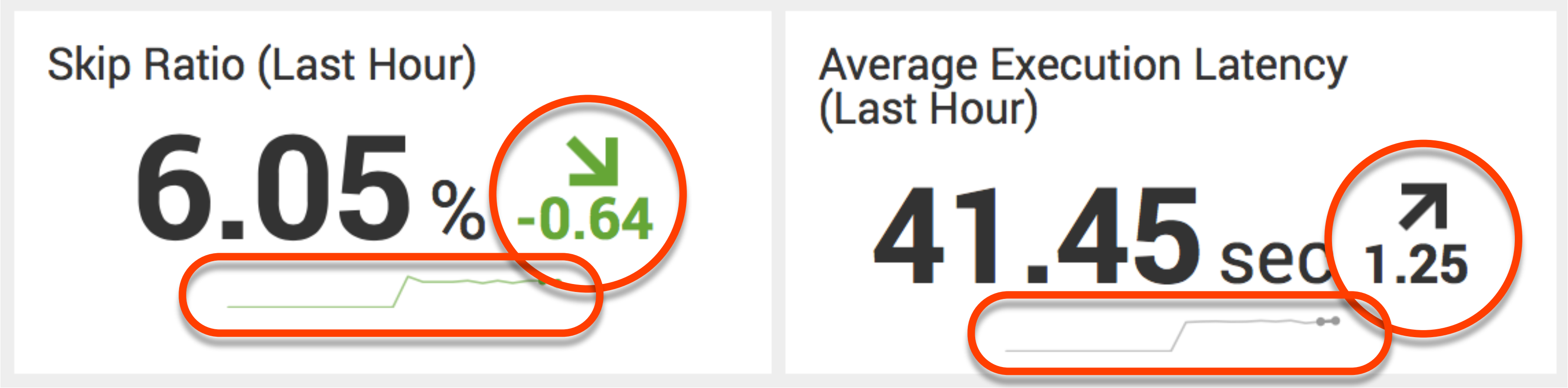
Distributed Management Console (DMC)

- 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*.



Skip Ratio (Last Hour)

6.05 % ↘ -0.64

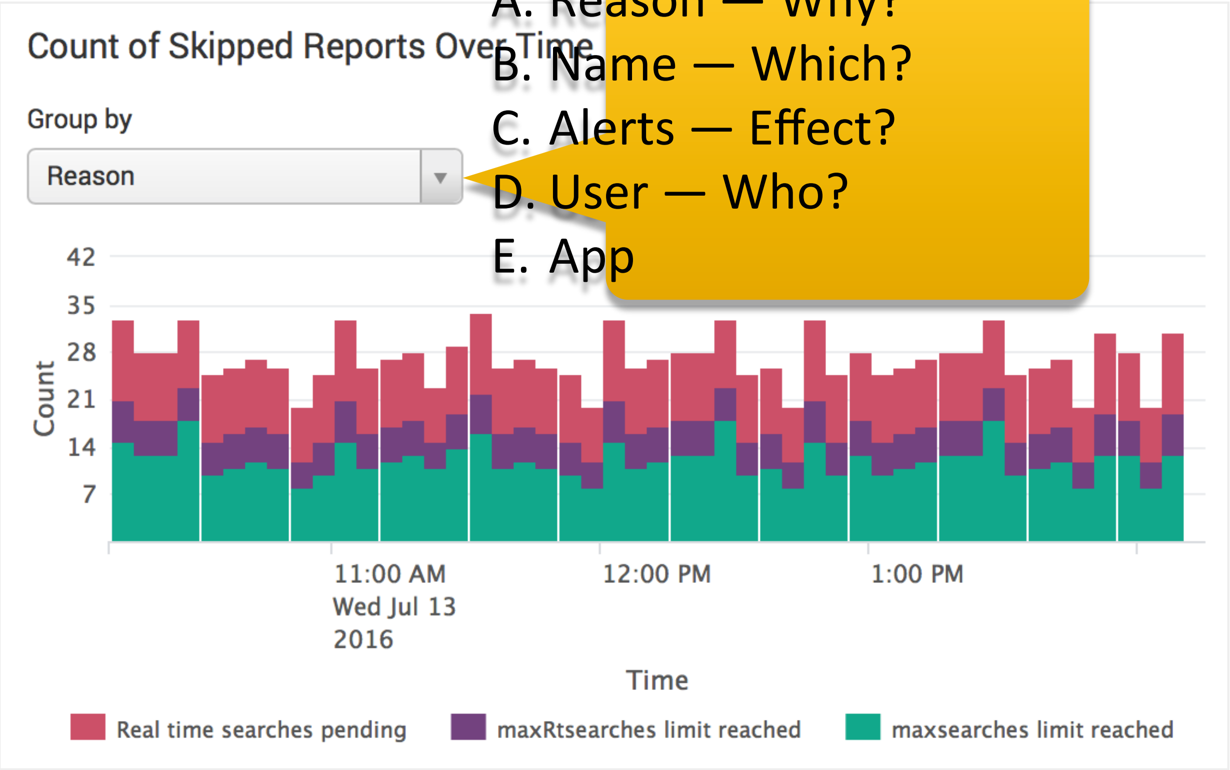
Average Execution Latency (Last Hour)

41.45 sec ↗ 1.25

DMC Scheduler Activity: Skipped Searches

What this chart shows:

Discretized counts of skipped searches.



Count of Skipped Reports Over Time

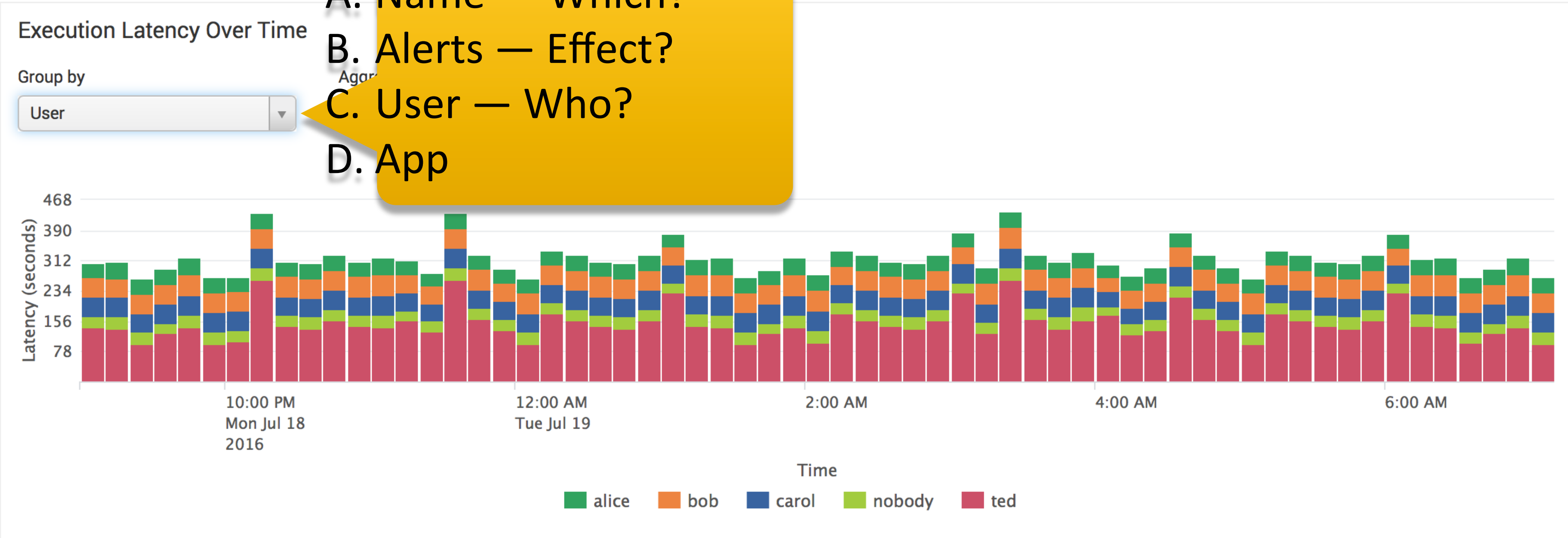
Group by

Reason ▾

42
35
28
21
14
7
Count
11:00 AM
Wed Jul 13
2016
12:00 PM
1:00 PM
Time
🟥 Real time searches pending
🟪 maxRtsearches limit reached
🟩 maxsearches limit reached

A. Reason — Why?
B. Name — Which?
C. Alerts — Effect?
D. User — Who?
E. App

DMC Scheduler Activity: Latency



A. Name — Which?

B. Alerts — Effect?

C. User — Who?

D. App

Execution Latency Over Time

Group by: User

What this chart shows: Discretized amounts of latency.

Takeaways

- Recent Splunk Enterprise versions added better *priority scoring* and *search windows* for much improved saved search scheduling by at least 25%.
- For infrequent searches (hourly, daily, etc.) use *schedule windows*, preferably *auto* windows.
- 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.
- 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.

THANK YOU

.conf2016

splunk>