Observations And Recommendations On Splunk Performance

Dritan Bitincka
Principal Architect, 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 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.
About Me

- Member of Splunk Tech Services
- >5 Years at Splunk
- Large scale and Cloud deployments
- 6th .conf
Agenda

- Performance & Bottlenecks
- **Understanding fundamentals:**
  - Indexing:
    - Index-time pipelines
    - Index testing
  - Searching:
    - Searching in *isolation* & under *indexing load*
    - Types of searches
    - Mixed workload impact on resources
Testing Disclaimers

• Testing on arbitrary datasets in a “closed course” (lab) environment
• Do not take out of context
Typical “my Splunk is not performing well” conversation

A: My Splunk is slow
B: Okay, so what exactly is slow?
A: I dunno, it just feels slow...maybe I’ll just get some SSDs
Splunk, like all distributed computing systems, has various bottlenecks that manifest themselves differently depending on workloads being processed.

- Winston Churchill
Identifying Performance Bottlenecks

- Understand data flows
  - Splunk operations pipelines
- Instrument
  - Capture metrics for relevant operations
- Run tests
- Draw conclusions
  - Chart and table metrics, looks for emerging patterns
- **Make recommendations**
Put That In Your Pipeline And Process It

Splunk data flows thru several such pipelines before it gets indexed
 Lots Of Pipelines

Parse Queue
- tcp/udp pipeline
- Tailing
- fifo pipeline
- fschange
- exec pipeline

Parsing Pipeline
- utf8
- linebreaker
- header

Merging Pipeline
- aggregator

Typing Pipeline
- Regex replacement
- annotator

Index Pipeline
- tcp out
- syslog out
- indexer

LINE_BREAKER
- TRUNCATE

SHOULD_LINEMERGE
- BREAK_ONLY_BEFORE
- MUST_BREAK_AFTER
- TIME_*

TRANSFORMS-xxx
- SEDCMD
- ANNOTATE_PUNCT
Index-time Processing

Event Breaking
- **LINE_BREAKER**: <where to break the stream>
- **SHOULD_LINEMERGE**: <enable/disable merging>

Timestamp Extraction
- **MAX_TIMESTAMP_LOOKAHEAD**: <# chars in to look for ts>
- **TIME_PREFIX**: <pattern before ts>
- **TIME_FORMAT**: <strptime format string to extract ts>

Typing
- **ANNOTATE_PUNCT**: <enable/disable punct:: extraction>
Testing: Dataset A

• 10M syslog-like events:
  
  08-24-2016 15:55:39.534 <syslog message >
  08-24-2016 15:55:40.921 <syslog message >
  08-24-2016 15:55:41.210 <syslog message >
  
• Push data thru:
  – **Parsing > Merging > Typing** Pipelines
    ‣ Skip **Indexing**
  – Tweak various props.conf settings

• Measure

---

MLA: \texttt{MAX\_TIMESTAMP\_LOOKAHEAD} = 24
LM: \texttt{SHOULD\_LINEMERGE} = false
TF: \texttt{TIME\_FORMAT} = \texttt{%m-%d-%Y %H:%M:%S.%3N}
DC: \texttt{DATETIME\_CONFIG} = \texttt{CURRENT}
Index-time Pipeline Results

- Default: 9.5s
- MLA: 8.6s
- LM+TF: 6.3s
- LM+DC: 5.8s

MLA: MAX_TIMESTAMP_LOOKAHEAD = 24
LM: SHOULD_LINEMERGE = false
TF: TIME_FORMAT = %m-%d-%Y %H:%M:%S.%3N
DC: DATETIME_CONFIG = CURRENT
• All pre-indexing pipelines are expensive at default settings.
  - Price of flexibility

• If you’re looking for performance, minimize generality

- LINE_BREAKER
- SHOULD_LINEMERGE
- MAX_TIMESTAMP_LOOKAHEAD
- TIME_PREFIX
- TIME_FORMAT
Next: Let’s Index A Dataset B

- Generate a much larger dataset (1TB)
  - High cardinality, ~380 Bytes/event, 2.9B events

- Forward to indexer as fast as possible
  - Indexer:
    - Linux 2.6.32 (CentOS);
    - 2x12 Xeon 2.30 GHz (HT enabled)
    - 8x300GB 15k RPM drives in RAID-0
  - No other load on the box

- Measure
Indexing: CPU And IO
Indexing Test Findings

- **CPU Utilization**
  - ~17.6% In this case, **4-5** Real CPU Cores

- **IO Utilization**
  - Characterized by both reads and writes but not as demanding as search.
    - Note the `splunk-optimize` process

- **Ingestion Rate**
  - **30MB/s**
  - “Speed of Light” – no search load present on the server
Index Pipeline Parallelization

- Splunk 6.3+ can maintain multiple independent pipelines sets
  - i.e. same as if each set was running on its own indexer
- If machine is under-utilized (CPU and I/O), you can configure the indexer to run 2 such sets
- Achieve roughly double the indexing throughput capacity
- Try not to set over 2
- Be mindful of associated resource consumption
Indexing Test Conclusions

- **Distribute** as much as you can – Splunk scales horizontally
  - Enable more pipelines but be aware of compute tradeoff

- **Tune** event **breaking** and **timestamping** attributes in props.conf whenever possible

- Faster disk (ex. SSDs) would not have necessarily improved indexing throughput by much

- Faster, but not more, CPUs would have improved indexing throughput (multiple pipelines would need more CPUs)
Next: Searching

- Real-life search workloads are extremely complex and very varied to be profiled correctly

- But, we can generate arbitrary workloads covering a wide spectrum of resource utilization and profile those instead. Actual profile will fall somewhere in between
Some preparatory steps here

Find buckets based on search timerange

For each bucket check tsidx for events that match LISPY and find rawdata offset

For each bucket read journal.gz at offsets supplied by previous step

Process events: rename, extract, report, kv, alias, eval, lookup, subsecond

Filter events to match the search string (+ eventtyping tagging)

Write temporary results to dispatch directory

Return progress to SH Splunk’d

Repeat until search completes
Some preparatory steps here

- Find buckets based on search timerange
- For each bucket, check tsidx for events that match LISPY and find rawdata offset
- For each bucket, read journal.gz at offsets supplied by previous step
- Process events: rename, extract, report, kv, alias, eval, lookup, subsecond
- Filter events to match the search string (+ eventtyping tagging)
- Write temporary results to dispatch directory

Repeat until search completes

IO

Return progress to SH Splunk'd
Search Pipeline Boundedness

Some preparatory steps here

Find buckets based on search timerange

For each bucket check tsidx for events that match LISPY and find rawdata offset

For each bucket read journal.gz at offsets supplied by previous step

Process events: st rename, extract, report, kv, alias, eval, lookup, subsecond

Filter events to match the search string (+ eventtyping tagging)

Write temporary results to dispatch directory

Repeat until search completes

IO

CPU + Memory

Return progress to SH Splunk’d
Search Types

- **Dense**
  - Characterized predominantly by returning *many events* per bucket
    
    \[\text{index=web} \mid \text{stats count by clientip}\]

- **Sparse**
  - Characterized predominantly by returning *some events per bucket*
    
    \[\text{index=web some_term} \mid \text{stats count by clientip}\]

- **Rare**
  - Characterized predominantly by returning *only a few* events per index
    
    \[\text{index=web url=onedomain*} \mid \text{stats count by clientip}\]
Okay, Let’s Test Some Searches

- Use our already indexed data
  - It contains many unique terms with predictable term density

- Search under several term densities and concurrencies
  - Term density: 1/100, 1/1M, 1/100M
  - Search Concurrency: 4 – 60
  - Searches:
    - Rare: over all 1TB dataset
    - Dense: over a preselected time range

- Repeat all of the above while under an indexing workload

- Measure
Dense Searches

CPU Utilization (%)

IO Wait (%)

Hitting 100% CPU at core# = concurrency
Indexing With Dense Searches

CPU Utilization (%)

Indexing Throughput (KB/s)

Search Duration (s)
Dense Searches Summary

- Dense workloads are CPU bound
- Dense workload completion times and indexing throughput both negatively affected while running simultaneously
- **Faster disk wont necessarily help as much here**
  - Majority of time in dense searches is spent in CPU decompressing rawdata + other SPL processing
- **Faster and more CPUs would have improved overall performance**
Rare Searches

CPU Utilization (%)

Reads/s (from sar)

IO Wait (%)
Indexing With Rare Searches

- CPU Utilization (%)
  - searching+indexing 1/100M
  - searching+indexing 1/1M

- Reads/s (from sar)
  - searching+indexing 1/100M
  - searching+indexing 1/1M

- IO Wait (%)
  - searching+indexing 1/100M
  - searching+indexing 1/1M
More Numbers

Indexing Throughput (KB/s)
- Indexing only
- Indexing + searching 1/100M
- Indexing + searching 1/1M

Search Duration (s)
- Searching 1/100M
- Searching 1/1M

Search Duration (s)
- Searching + indexing 1/100M
- Searching + indexing 1/1M
Rare Searches Summary

- Rare workloads (investigative, ad-hoc) are IO bound
- Rare workload completion times and indexing throughput both negatively affected while running simultaneously
- 1/100M searches have a lesser impact on IO than 1/1M
- When indexing is on, in 1/1M case search duration increases substantially more vs. 1/100M. Search and indexing are both contenting for IO
- In case of 1/100M, **bloomfilters** help improve search performance
  - Bloomfilters are special data structures that indicate with 100% certainty that a term does not exist in a bucket (indicating to the search process to skip that bucket)
- Faster disks would have definitely helped here
- More CPUs would not have improved performance by much
Is My Search CPU Or IO Bound?

Guideline in absence of full instrumentation

- `command.search.rawdata` ~ CPU Bound
  - Others: .kv, .typer, .calcfields,
- `command.search.index` ~ IO Bound
Top Takeways/Re-Cap

- **Indexing**
  - Distribute – Splunk scales horizontally
  - Tune event breaking and timestamp extraction
  - Faster CPUs will help with indexing performance

- **Searching**
  - Distribute – Splunk scales horizontally
  - Dense Search Workloads
    - CPU Bound, better with indexing than rare workloads
    - Faster and more CPUs will help
  - Rare Search Workloads
    - IO Bound, not that great with indexing
    - Bloomfilters help significantly
    - Faster disks will help

- **Performance**
  - Avoid generality, optimize for expected case and add hardware whenever you can
Testing Disclaimer Reminder

1. Testing conducted on arbitrary datasets
2. "closed course" (lab) environment
3. Not to be interpreted out of context
Q & A

Feedback: dritan@splunk.com

You May Also Like

Search: Under the Hood
Worst Practices... and How to Fix Them
Splunk Performance Reloaded
THANK YOU

.conf2016