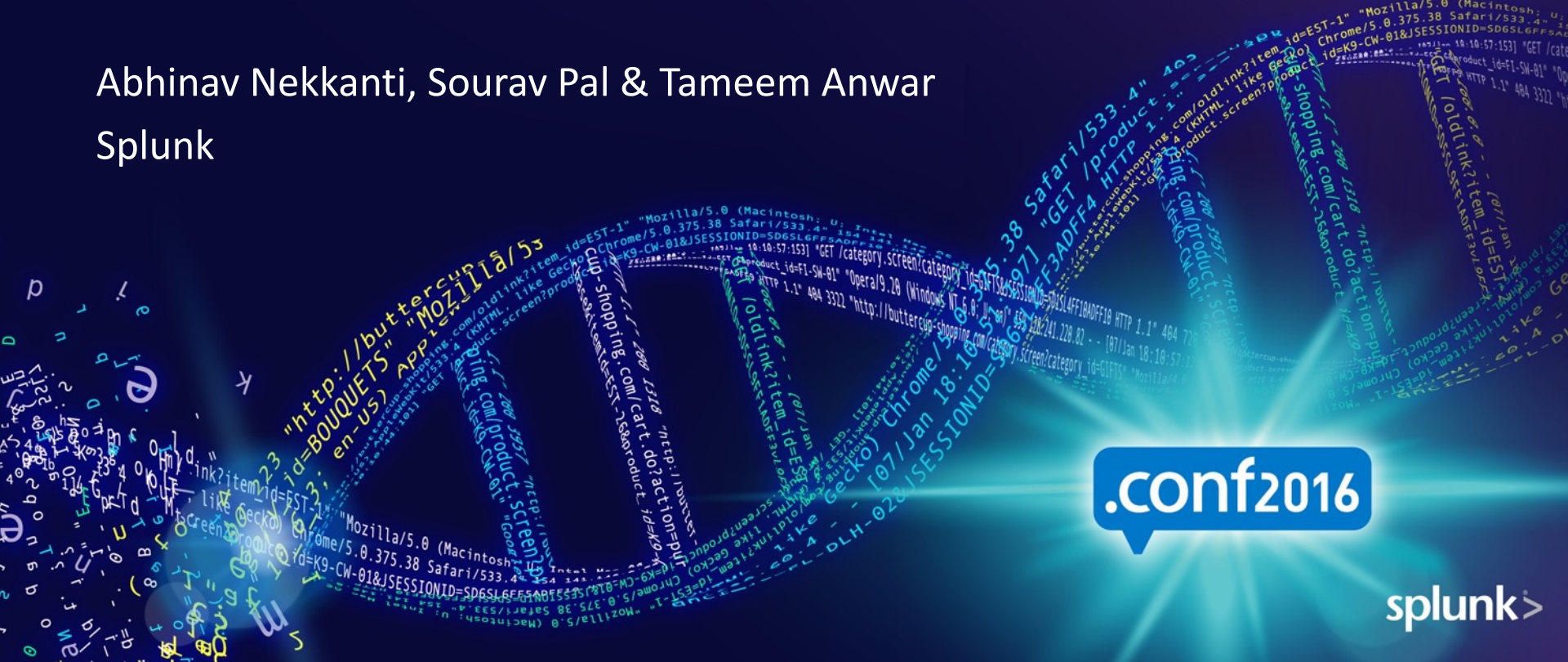
Harnessing Performance and Scalability with Parallelization

Abhinav Nekkanti, Sourav Pal & Tameem Anwar

Splunk

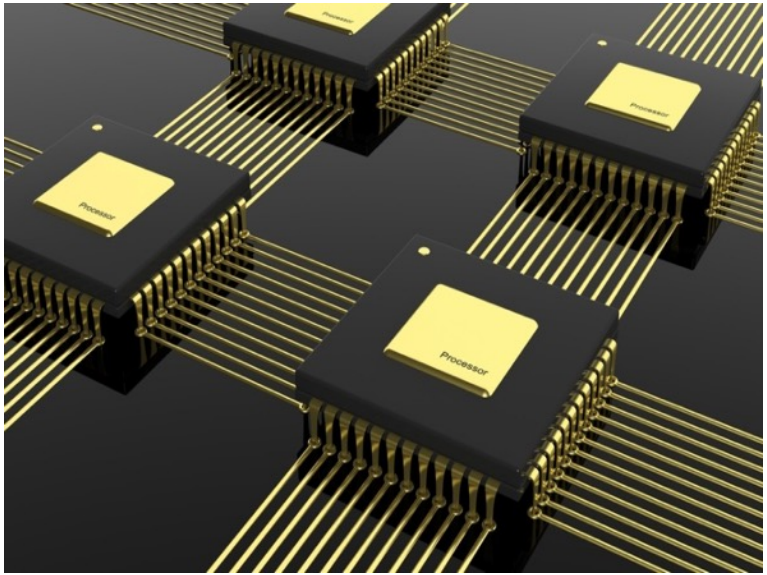


Disclaimer

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.

Agenda

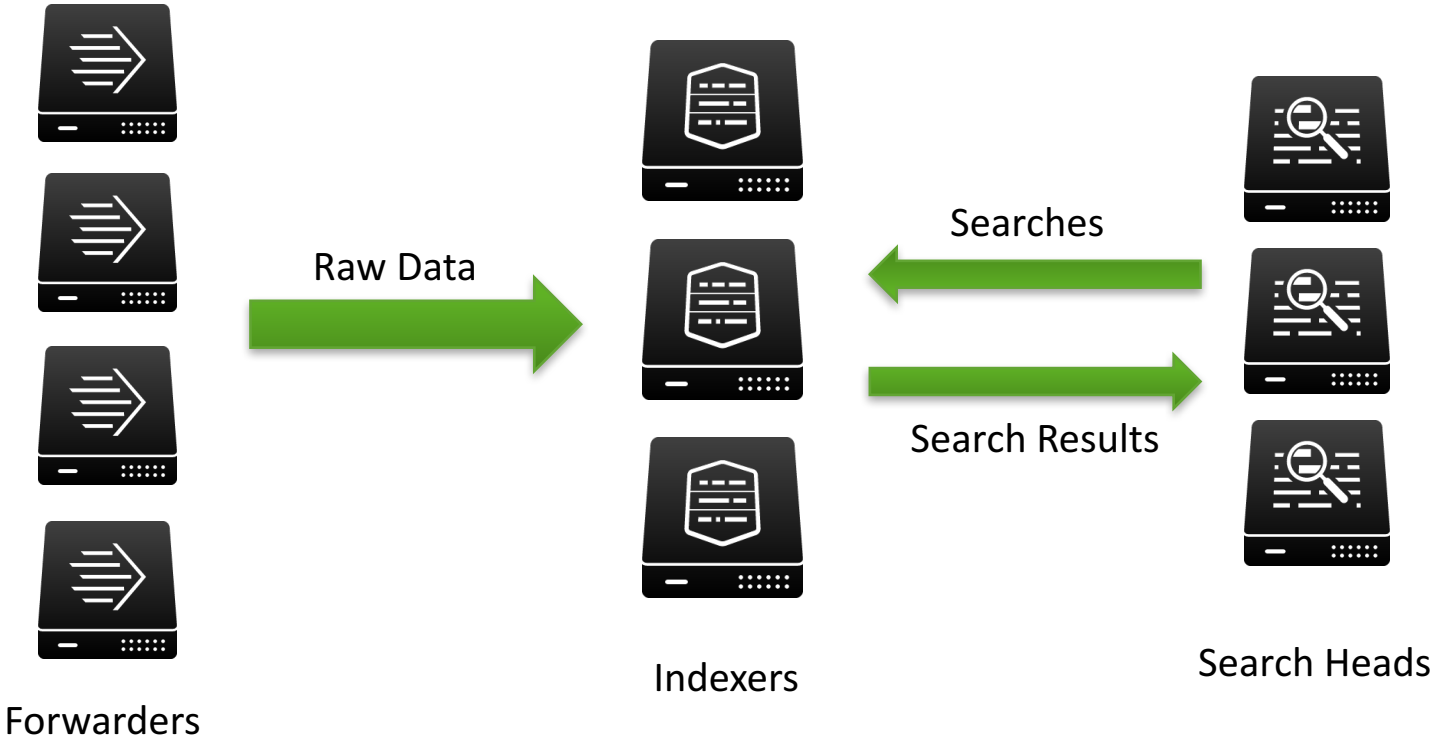
- Under-utilized hardware?
- Multiple ingestion pipelines
- Parallelizing search
- Performance
- Best practices



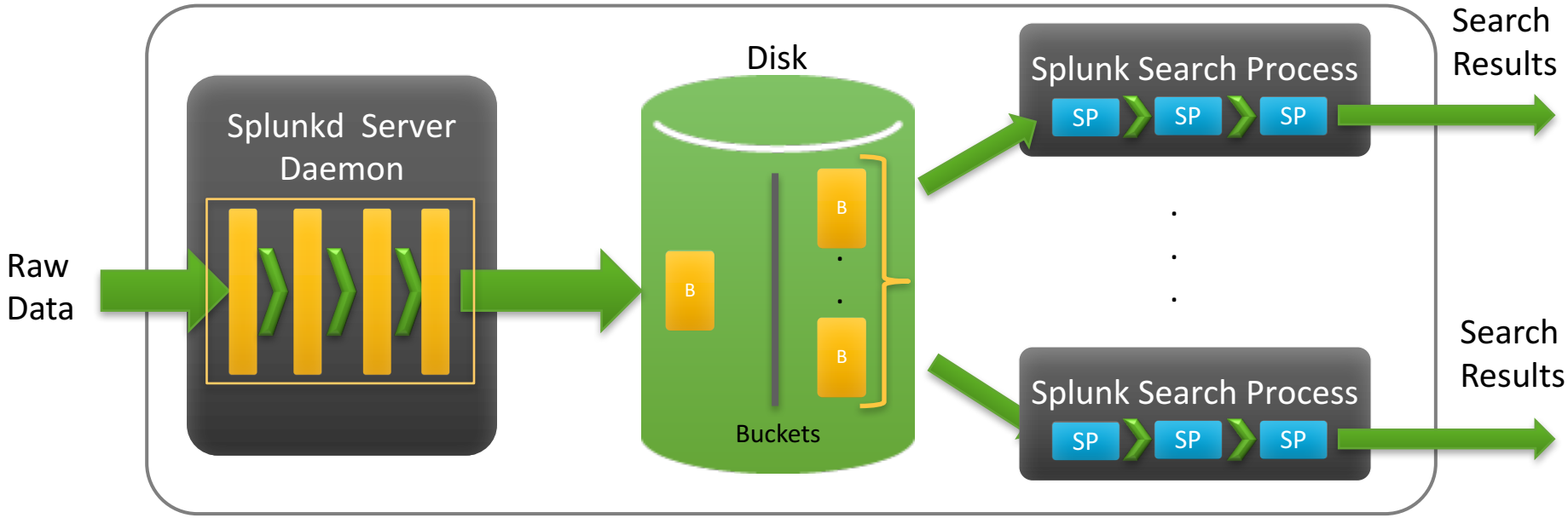
About Us

- Abhinav Nekkanti - Sr. Software Engineer, Splunk
  - Ingestion Pipeline
- Sourav Pal - Principal Engineer, Splunk
  - Search Parallelization
- Tameem Anwar - Software Engineer, Splunk
  - Performance

3 Tier Architecture

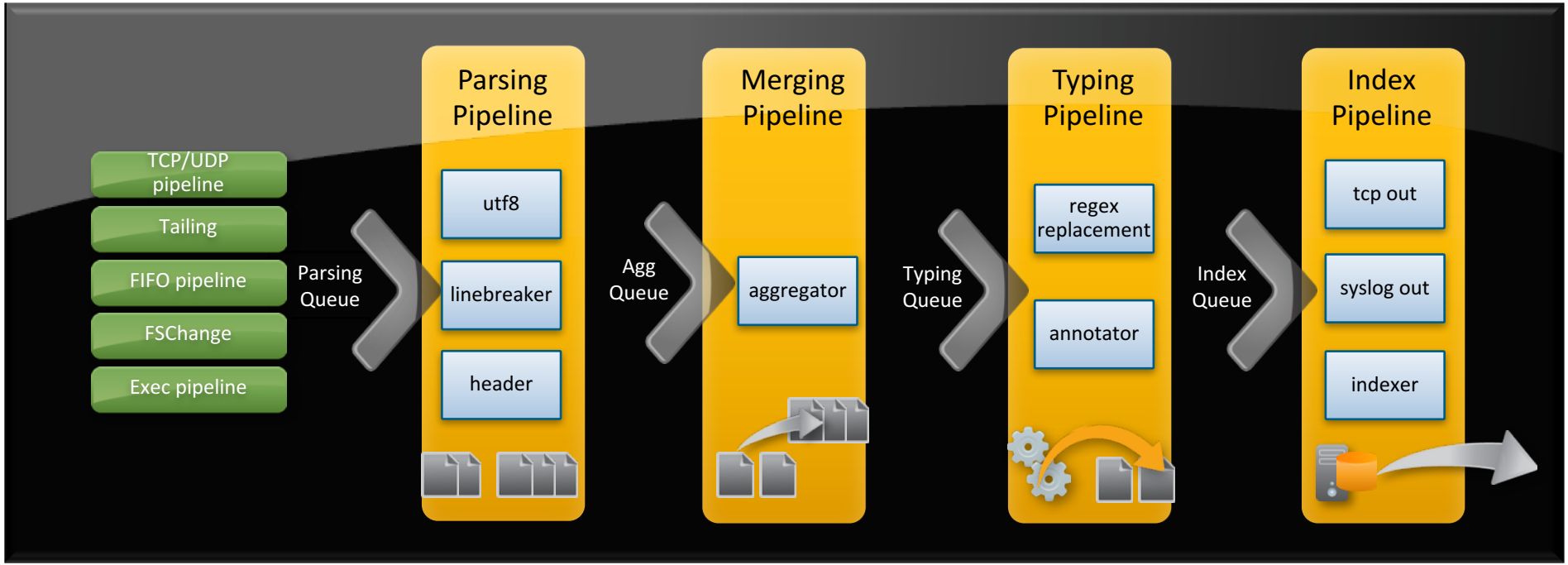


Insight into the Indexer



Traditional Indexer Hosts

Splunkd Server Daemon / Pipelineset



Ingestion Pipeline Set

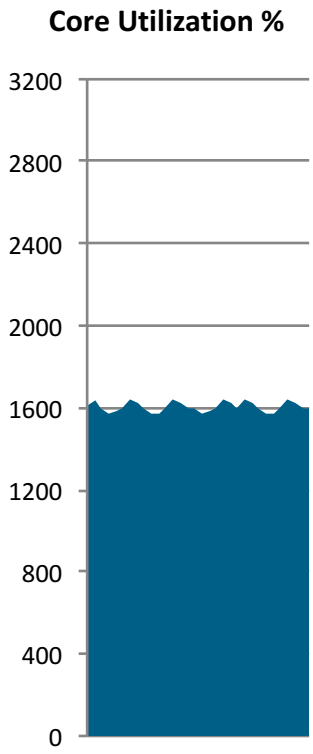
Indexer Core Utilization

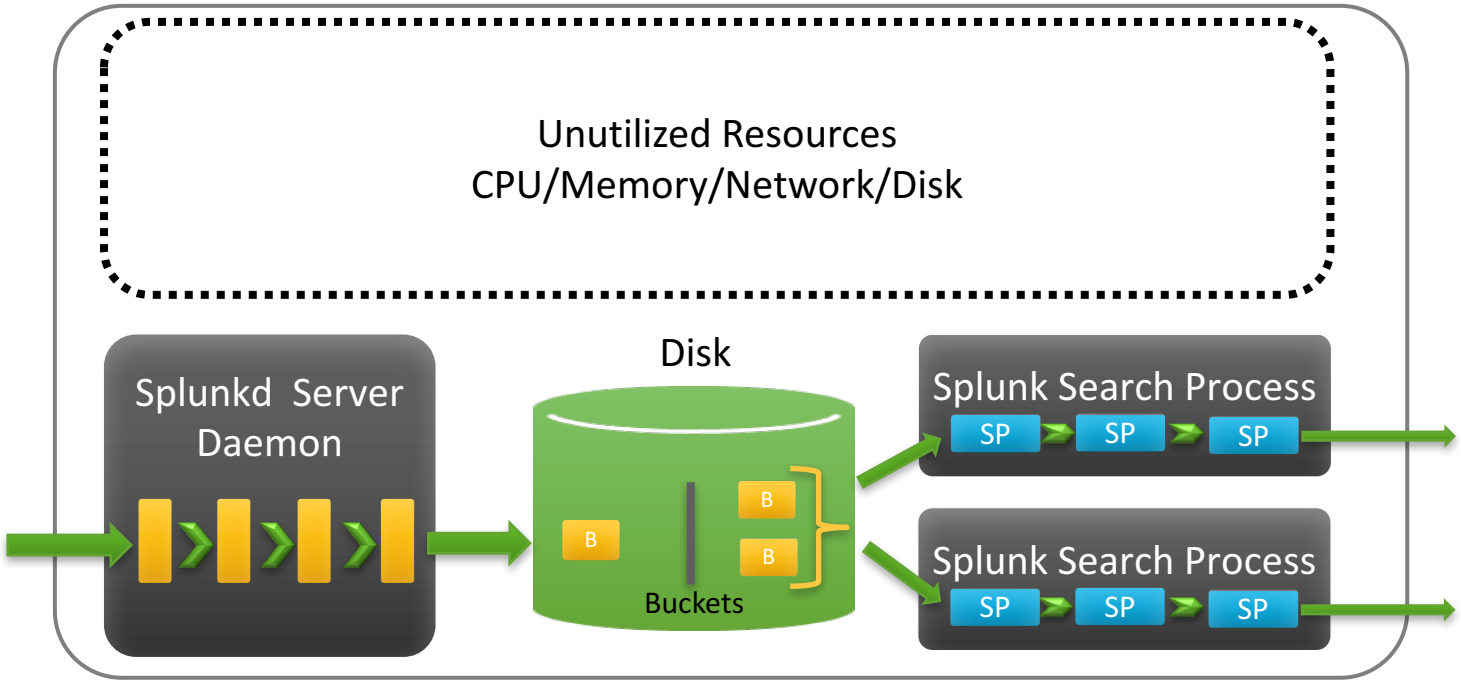
- Rule of Thumb:

Process	Cores (approx.)
Splunkd Server Daemon	4 to 6 cores
Splunk Search Process	1 core / search process

- Example core utilization of a Indexer Host:
  - 4 to 6 cores for Splunkd Server daemon
  - 10 X 1 cores for Splunk Search Processes
  - Total cores used: 14 to 16 cores

Under-utilized Indexer



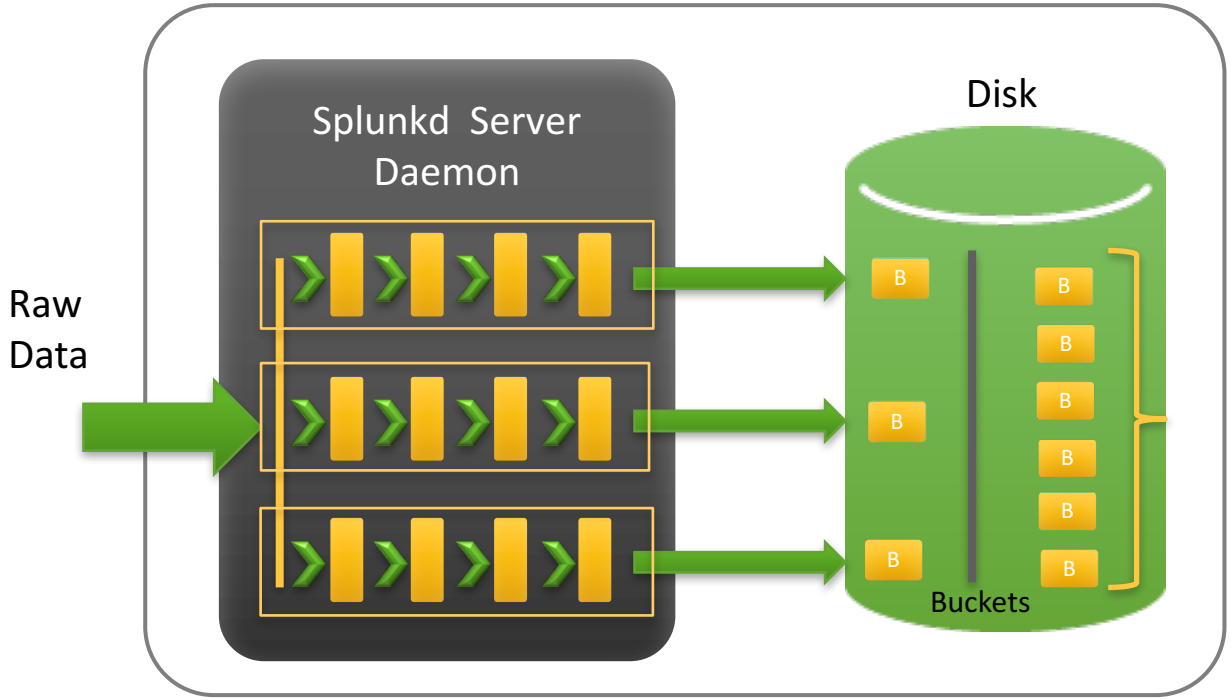


Performance Enhancements

- Multiple Pipeline Sets
  - Parallel ingesting pipeline sets
  - Improves resource utilization of the host machine
- Search Improvements
  - Faster batch searches using parallel search pipelines
  - Scheduler improvements
  - Faster Summary buildup

Multiple Ingestion Pipeline Sets

Splunkd with Multiple Ingestion Pipeline Sets



Indexer with 3 Pipeline Sets

Configuring Multiple Ingestion Pipeline Sets

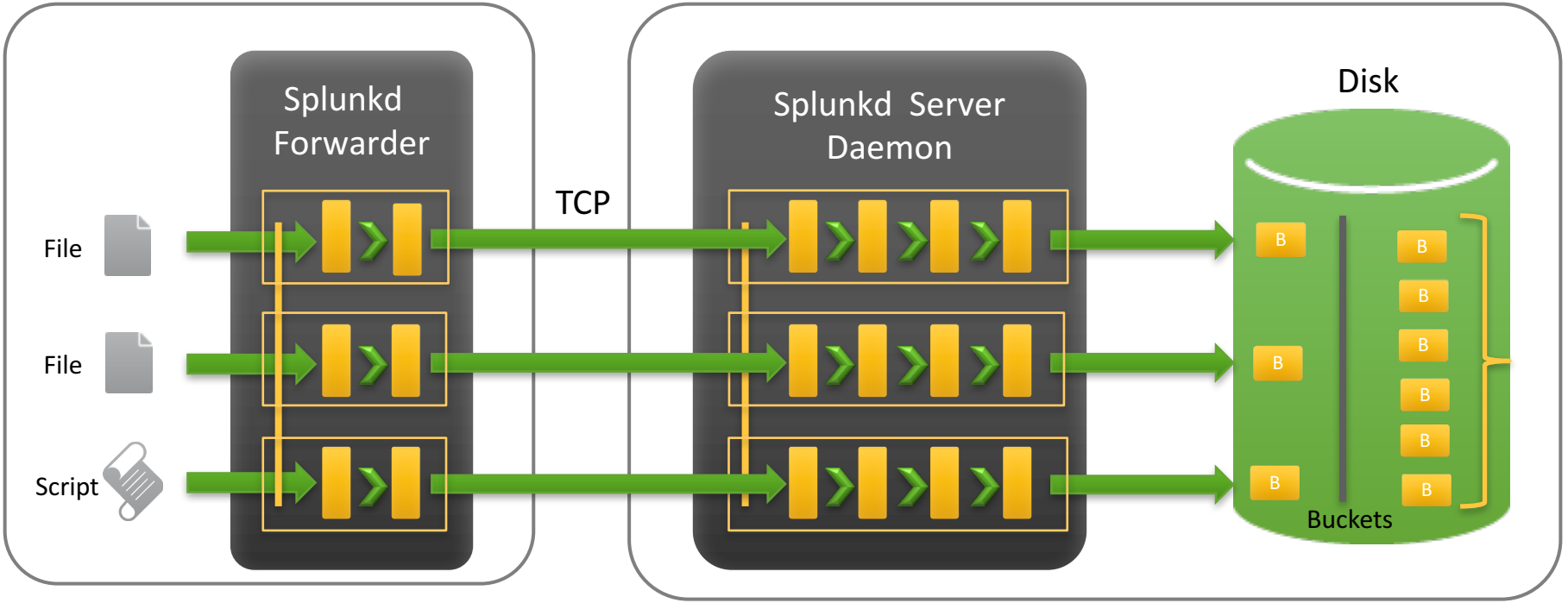
- $SPLUNK_HOME/etc/system/local/server.conf

```
[general]
parallelIngestionPipelines = 2
```

Multiple Ingestion Pipeline Sets – Details

- Each Pipeline Set has its own set of Queues, Pipelines and Processors
  - Exceptions are Input Pipelines which are usually singleton
- No state is shared across Pipeline Sets
- Data from a unique source is handled by only one Pipeline Set at a time

Multiple Ingestion Pipeline Sets over Network



Forwarder with 3 Pipeline Sets

Indexer with 3 Pipeline Sets

Multiple Ingestion Pipeline Sets – Monitor Input

- Each Pipelineset has its own set of TailReader, BatchReader and Archive Processor
- Enables parallel reading of files and archives on Forwarders
- Each file/archive is assigned to one pipeline set

Multiple Ingestion Pipeline Sets - Forwarding

- Forwarder:
  - One tcp output processor per pipeline set
  - Multiple tcp connections from the forwarder to different indexers at the same time
  - Load balancing rules applied to each pipeline set independently
- Indexer:
  - Every incoming tcp forwarder connection is bound to one pipeline set on the Indexer

Multiple Ingestion Pipeline Sets - Indexing

- Every pipeline set will independently write new data to indexes
- Data is written in parallel to better utilize resources
- Buckets produced by different pipeline sets could have overlapping time ranges

Search :
Parallelization Efforts
Performance Improvements

Search Parallelization: Performance Improvement

Splunk Searches are faster.

- Parallelizing the Search Pipeline
- Improving the Search Scheduler
- The Summary Building is parallelized and faster.

Search Pipeline

Batch Search

Reading Order

Reading Order

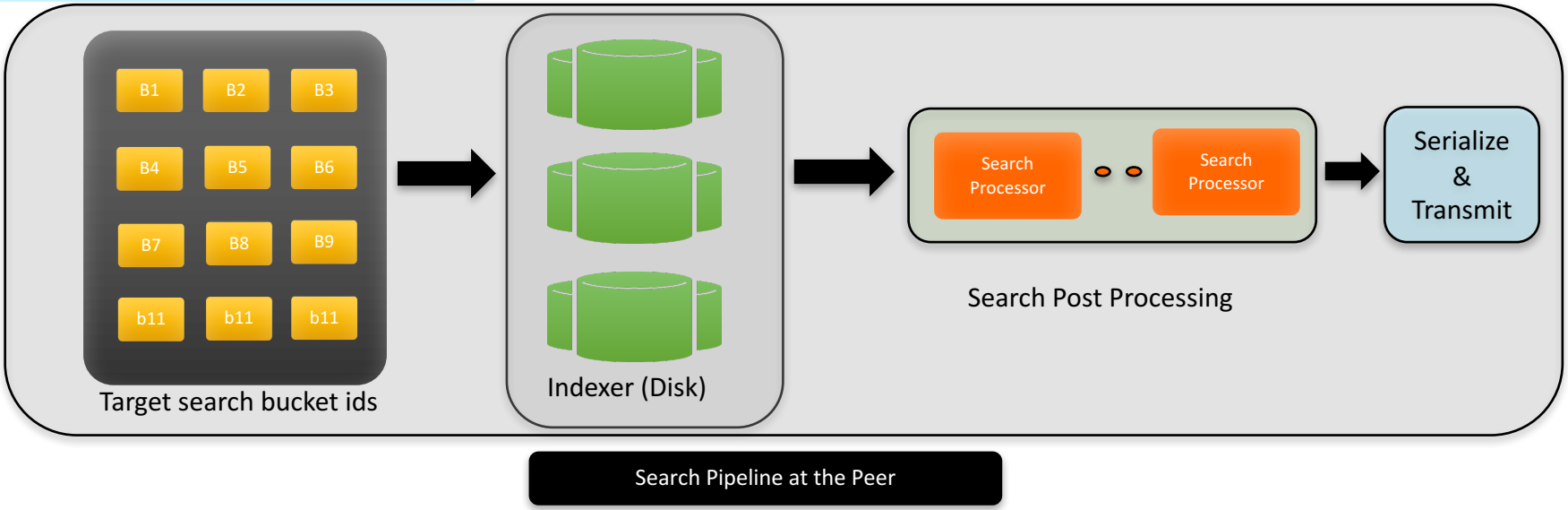
Option1:…B3 B5 B1 B2 B1 B6

Option2:…B6 B5 B4 B3 B2 B1

Option 3…B6 B5 B4 B7 B4 B9

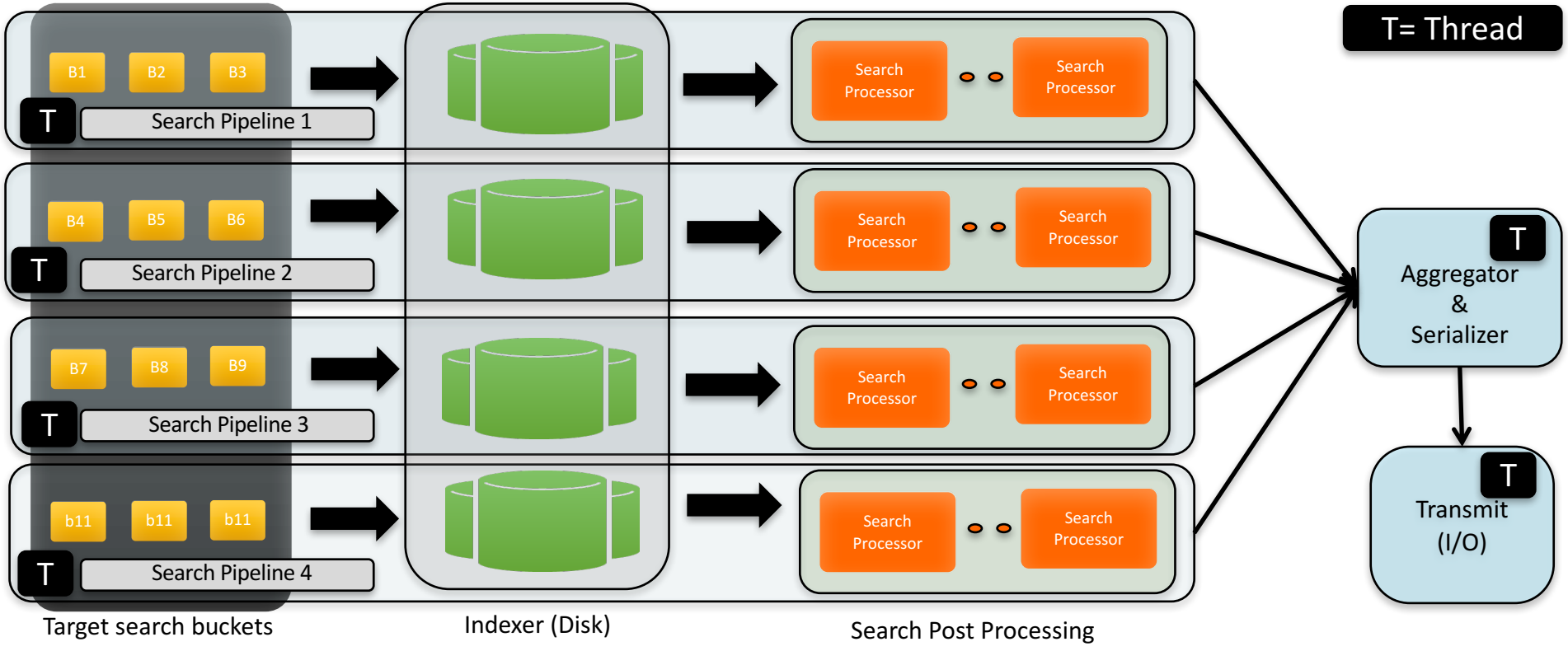
- Cursored Search: Time ordered data retrieval
- Batch Search: Bucket ordered data retrieval

Facilitates parallel processing of buckets independently across multiple pipeline



Search Pipeline at the Peer

Batch Search: Pipeline Parallelization



Batch Search: Pipeline Parallelization

- Under-utilized indexers provide us opportunity to execute multiple search pipelines.
- Batch Search time-unordered data access mode is ideal for multiple search pipelines.
- No state is shared i.e. no dependency exists across Search Pipelines.
- Peer/Indexer side optimizations.
- Takeaway :
  - Under utilized indexers are candidates for search pipeline parallelization.
  - Do NOT enable if indexers are loaded.

Configuring the Batch Search in Parallel mode

- How to enable?

  $SPLUNK_HOME/etc/system/local/limits.conf

```
[search]
batch_search_max_pipeline  = 2
```

- What to expect?

  Search performance in terms of retrieving search results improved.

  Increase in number of threads

Search Scheduler Improvements

- Scheduler improvements in Splunk Enterprise:
  - Priority Scoring
  - Schedule Windows
- Performance improvements over previous schedulers
  - Lower Lag
  - Fewer skipped searches

Search Scheduler Improvements
Priority Score

Problem:

Simple single-term priority scoring could result in saved search lag, skipping, and starvation (under CPU constraint).

Solution:

Better multi-term priority scoring mitigates problems and improves performance by 25%.

```
score(j)    = next_runtime(j)
            + average_runtime(j) ×
priority_runtime_factor
            – skipped_count(j) × period(j) ×
              priority_skipped_factor
            + schedule_window_adjustment(j)
```

Search Scheduler Improvements

Problem :

Scheduler can not 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 lag or skipping.

Solution :

Give a *schedule window* to searches that don’t have to run at specific 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.

- A search with a window helps *other* searches.
- Search windows *should not* be used for searches that run every minute.
- Search windows *must* be less than a search’s period

Configuring Search Scheduler

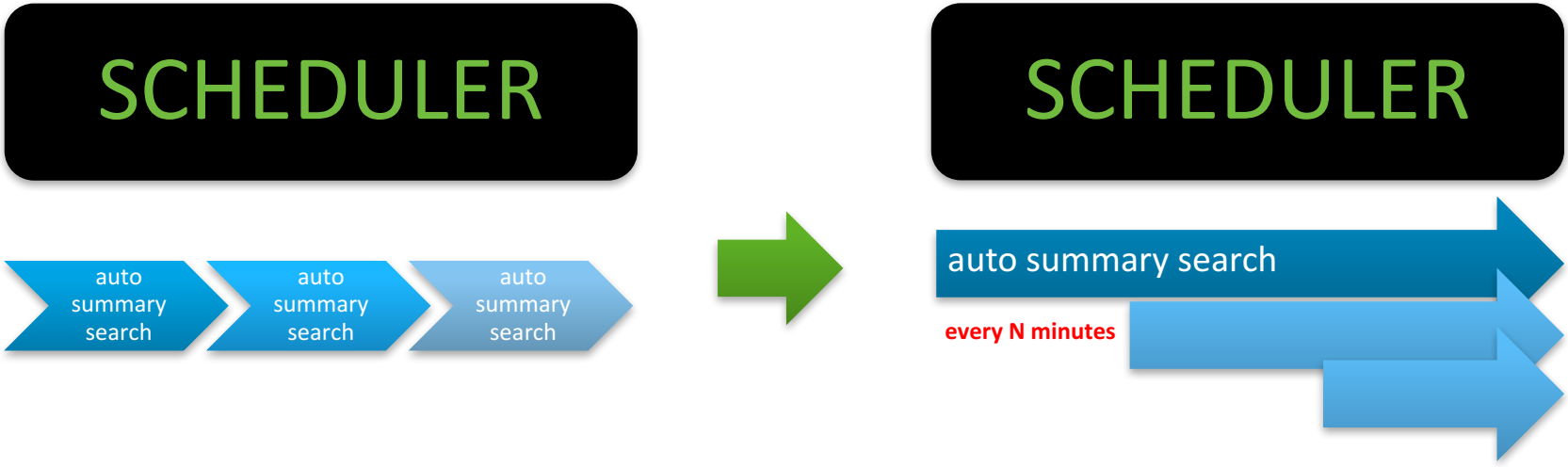
$SPLUNK_HOME/etc/system/local/limits.conf

```
[scheduler]
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 * * *
```

Search: Parallel Summarization

- Sequential nature of building summary data for data model and saved reports is slow.
- Summary Building process has been parallelized.

Summary Building Parallelization



Sequential Summary Building

Parallelized Summary Building

Configuring Summary Building for Parallelization

- $SPLUNK_HOME/etc/system/local/savedsearches.conf

```
[default]
auto_summarize.max_concurrent = 1
```

- $SPLUNK_HOME/etc/system/local/datamodels.conf

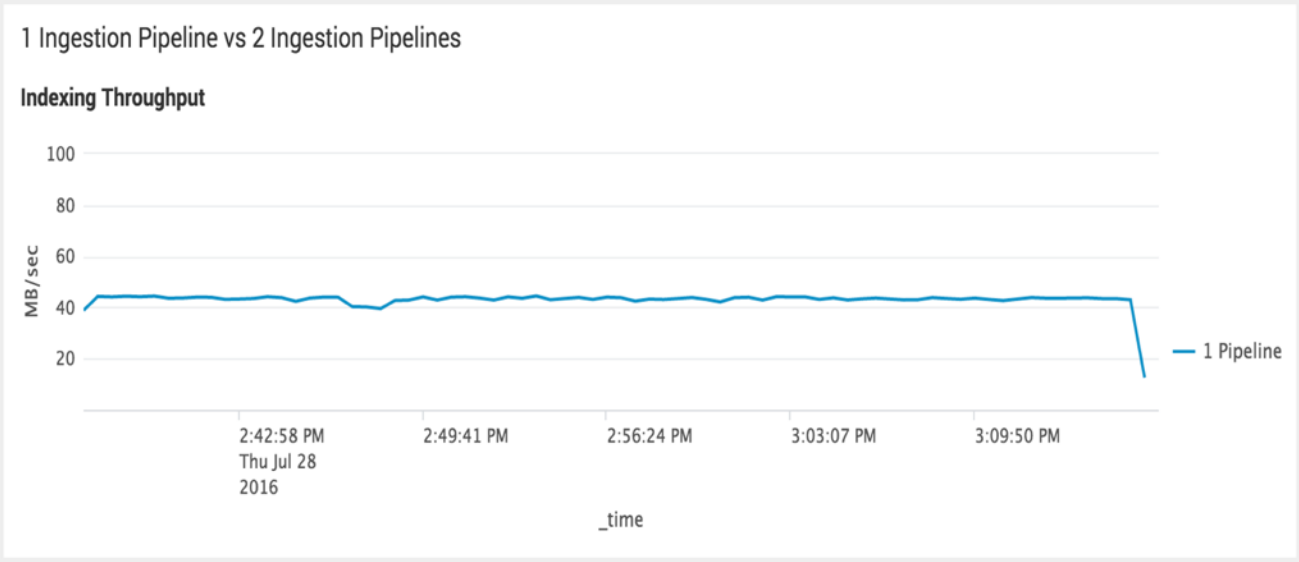
```
[default]
acceleration.max_concurrent = 2
```

Performance

Performance Tests

- System Info
  - 2x12 Xeon 2.30 GHz
  - 24 cores (48 w/HT)
  - 64 GB RAM
  - 8 x 300GB 15k RPM disks in RAID-0
  - 1 Gb Ethernet NIC
  - CentOS 7.6
- No other load on the box

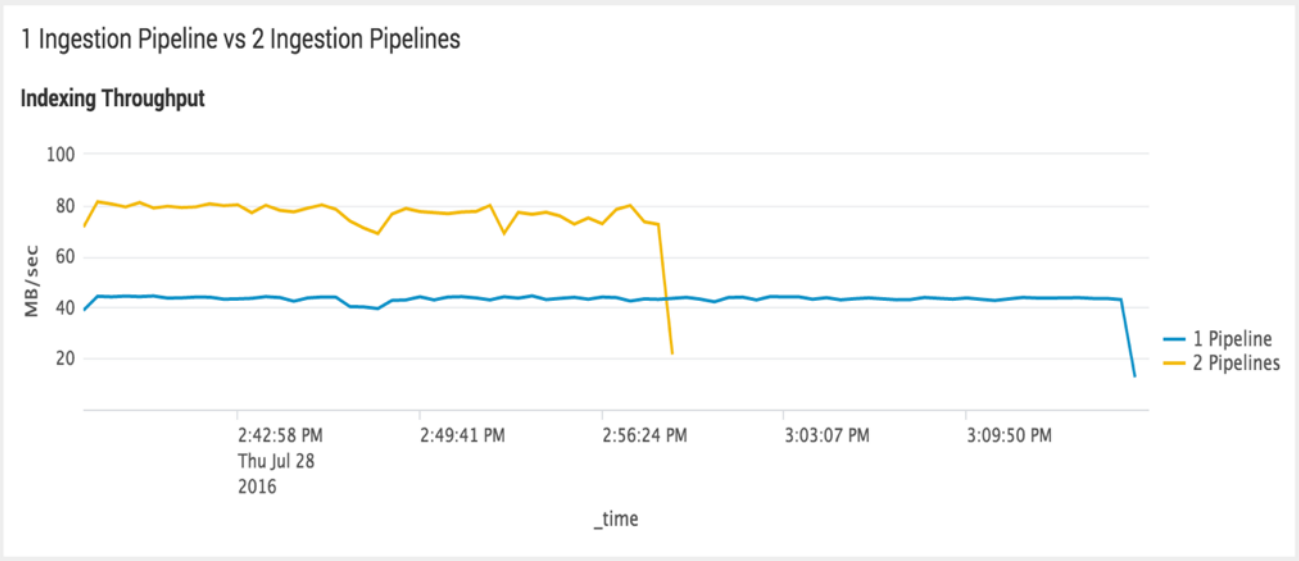
Indexing



Pipelines	Time taken (minutes)
1	40.25 m
2	

- Index a 100 GB generic syslog dataset. No search loads.
- Average Indexing Throughput – 41.40 MB/s

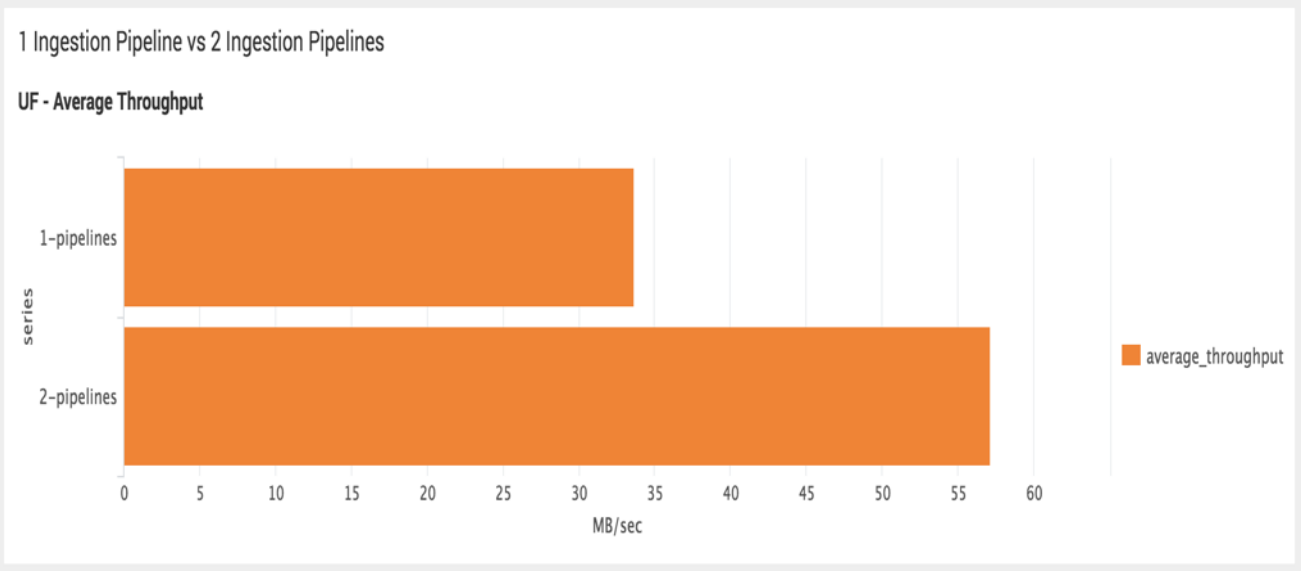
Indexing



Pipelines	Time taken (minutes)
1	40.25 m
2	21.16 m

- Average Indexing Throughput – 78.80 MB/s
- 90 % Increase in Average Indexing Throughput
- On an average Splunk utilized 2x CPU cores , 1.3x Memory and 2x Disk IOPS

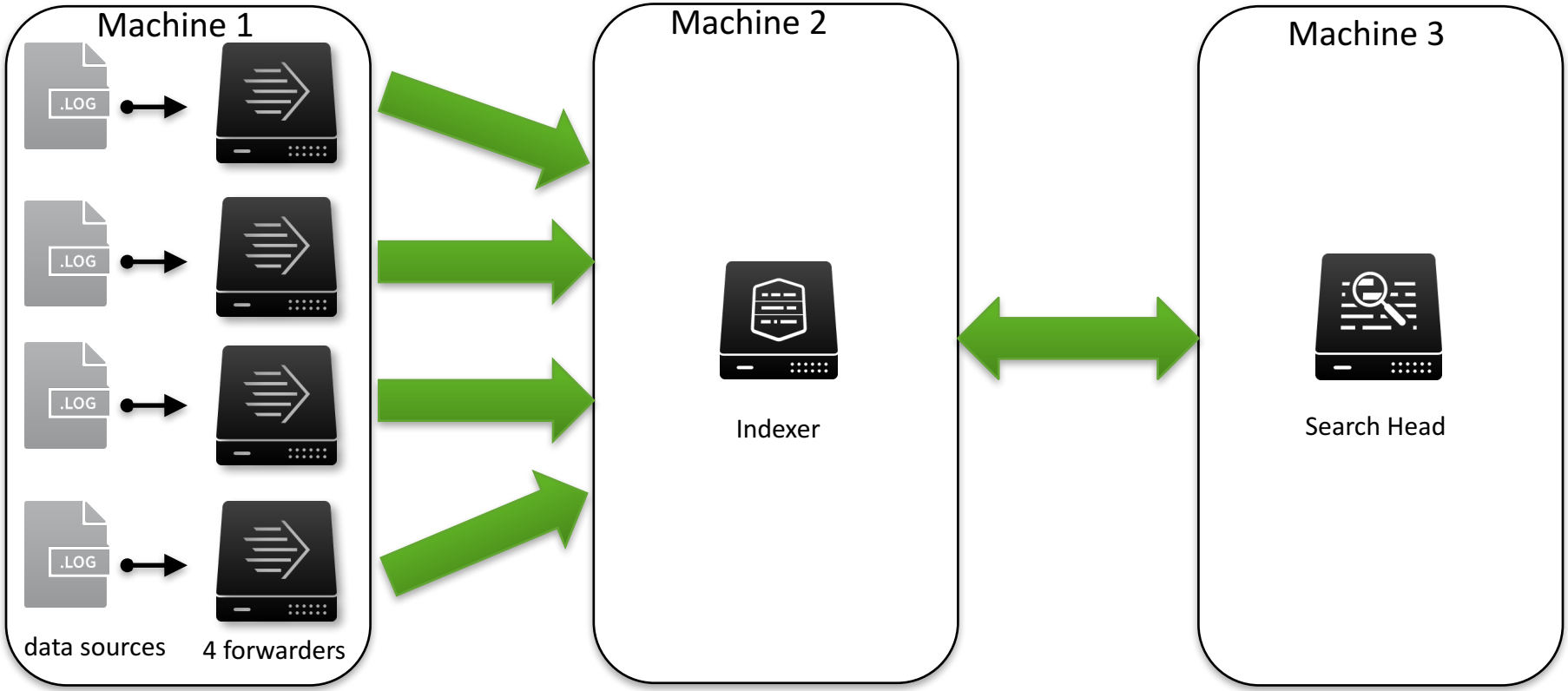
Forwarding



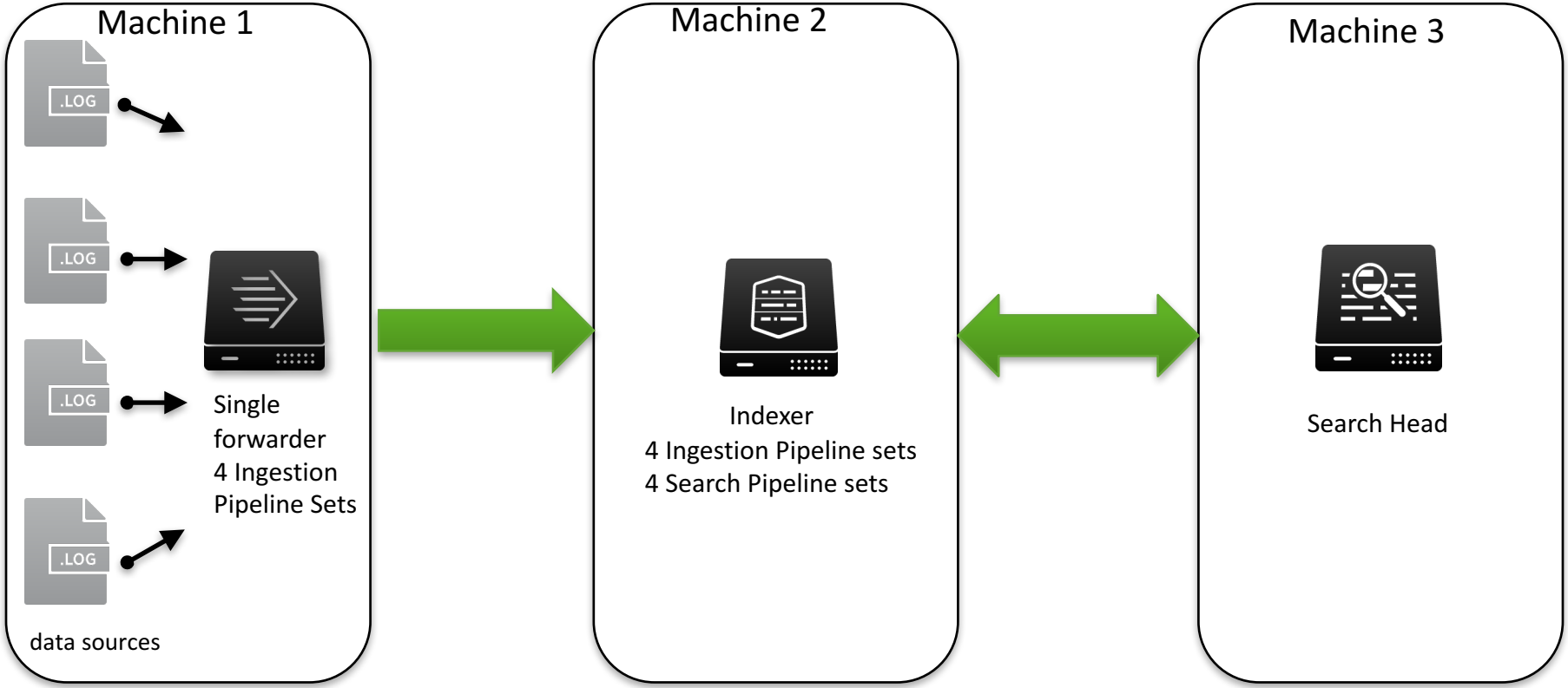
Pipelines	Average Throughput
1	33.6 MB/s
2	57.1 MB/s

- UF sending 100 GB syslog dataset (1k files)
- 70 % Increase in Average Throughput
- On an average Splunk utilized 2x the resources

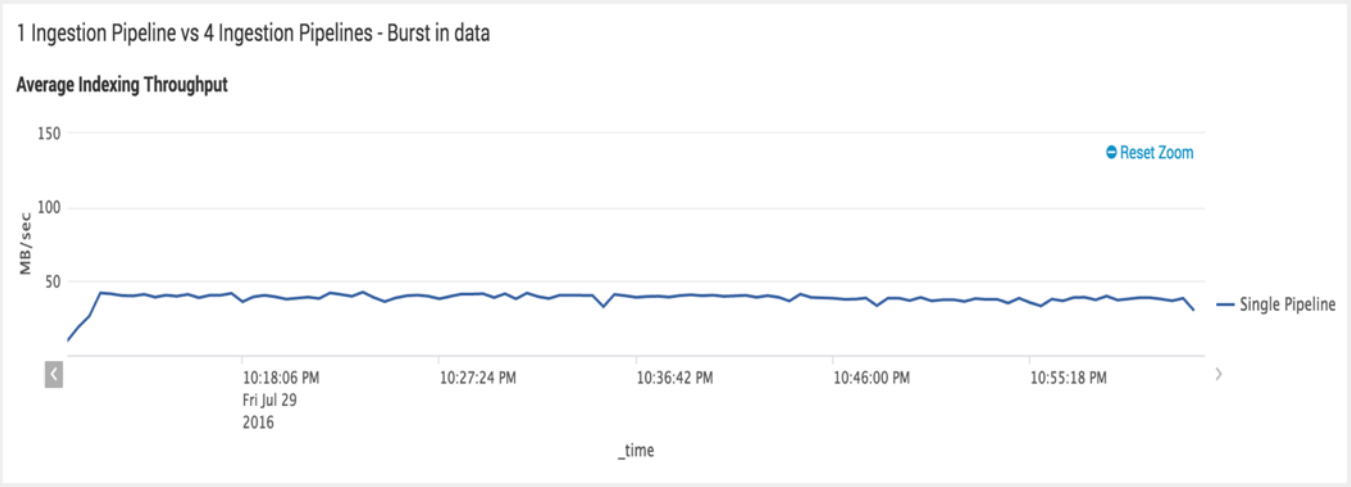
Splunk without Parallelization



Splunk with Parallelization



Burst in Indexing Load + Searches

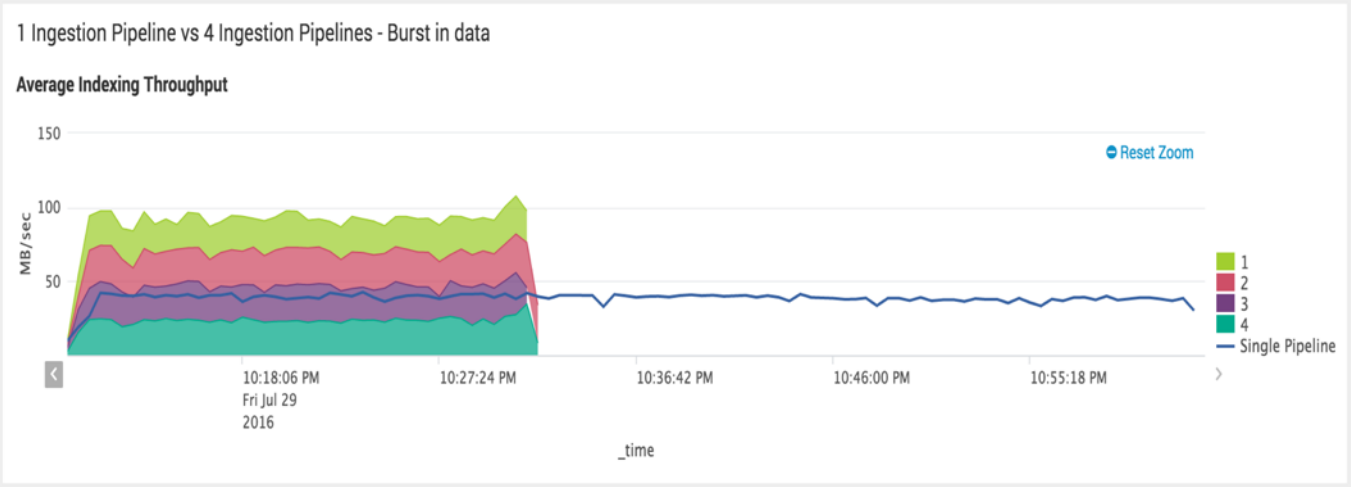


Ingestion Pipelines	Time (mins)
1	53 m
4	

Splunk without Parallelization

- Data forwarded @ 10 MB/s + Monitor 100 GB dataset
- Average Indexing Throughput – 39.12 MB/s
- Number of Concurrent Searches – 4

Burst in Indexing Load + Searches

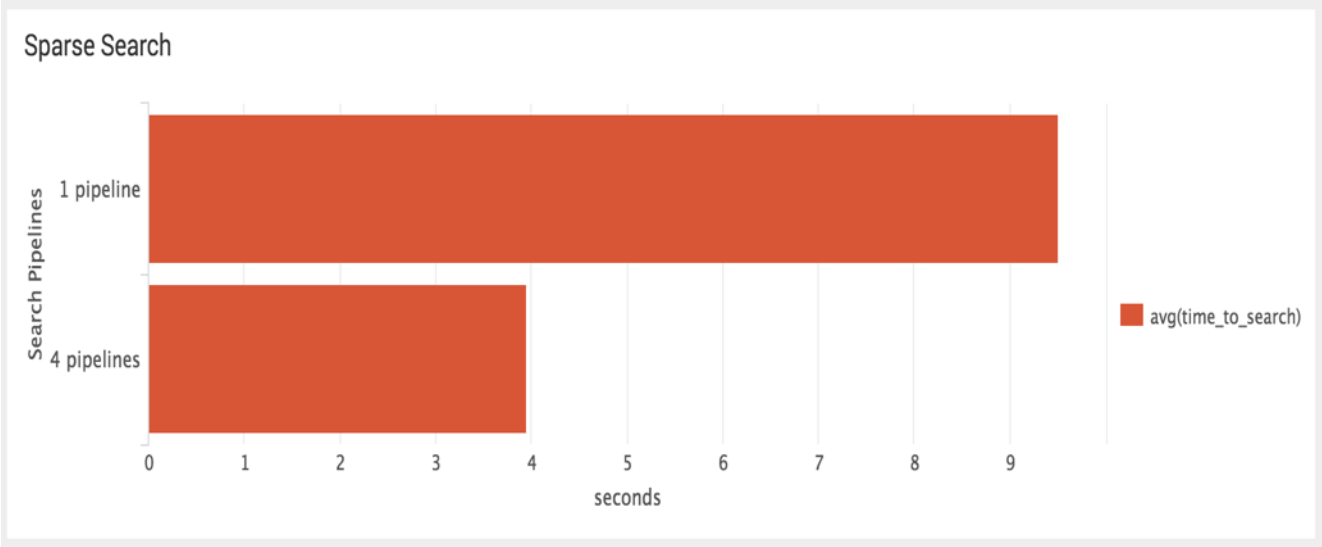


Ingestion Pipelines	Time (mins)
1	53 m
4	22.5 m

Splunk with Parallelization

- Data forwarded @ 10 MB/s + Monitor 100 GB dataset
- Average Indexing Throughput – 94.7 MB/s
- 142% Increase in Average Indexing Throughput
- Number of Concurrent Searches – 4

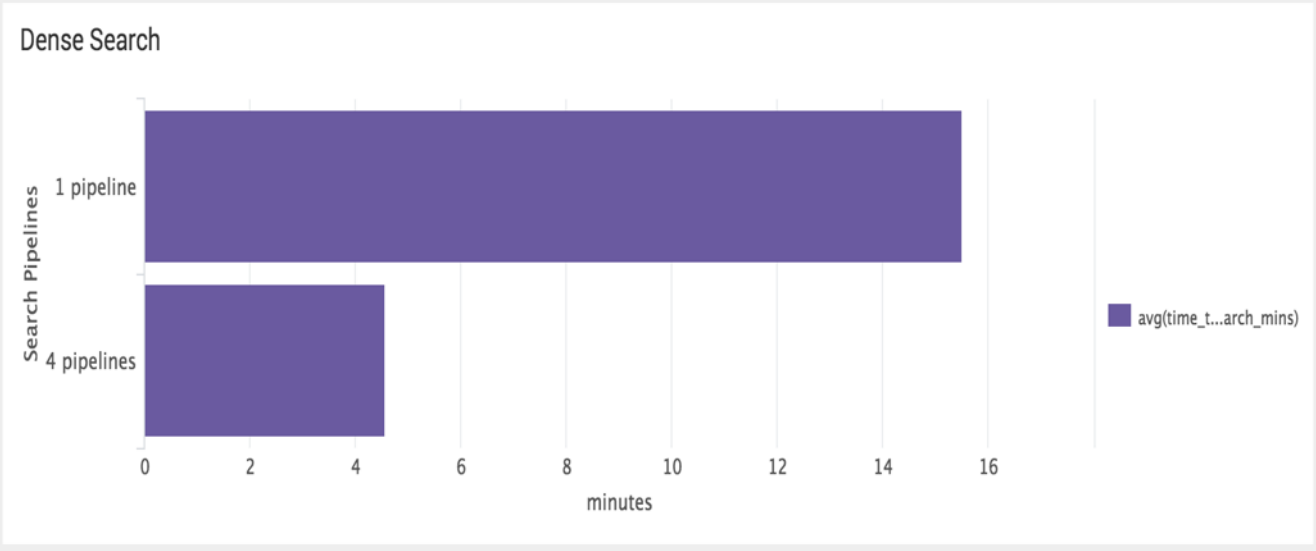
Batch Mode Sparse Search



Search Pipelines	Time (seconds)
1	9.51 s
4	3.90 s

- Sparse Search – Characterized predominately by returning some events per bucket
- 1 Search Pipeline vs 4 Search Pipelines
- Search is 2.4x faster with Search Parallelization

Batch Mode Dense Search



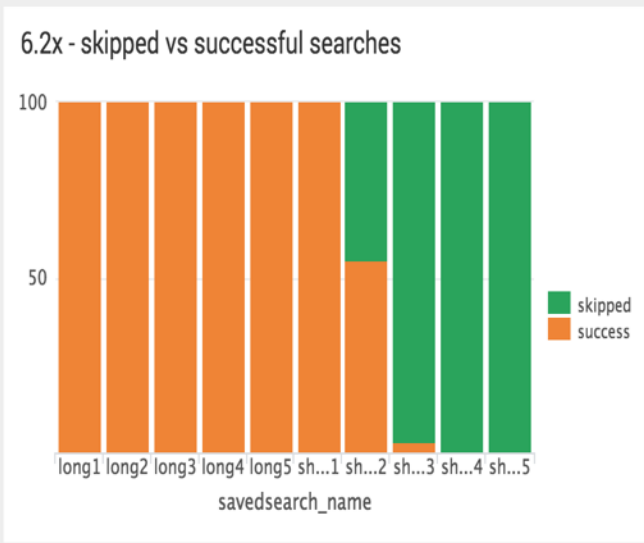
Search Pipelines	Time (minutes)
1	15.5 m
4	4.57 m

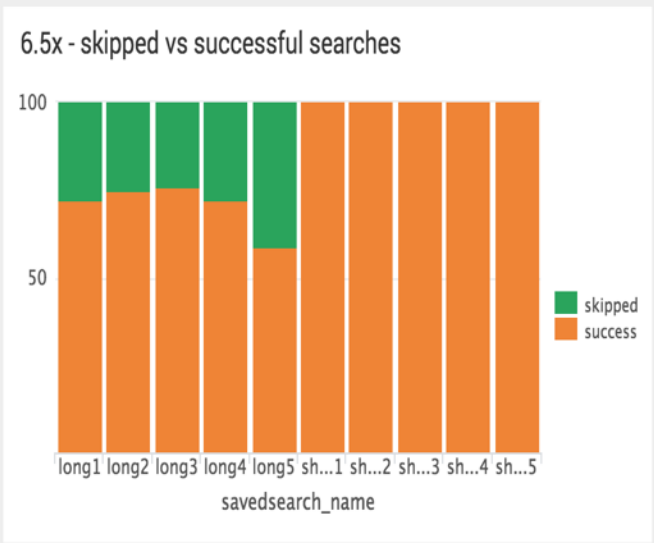
- Dense Search – Characterized predominately by returning many events per bucket
- 1 Search Pipelines vs 4 Search Pipelines
- Search is 3.4x faster with Search Parallelization

Scheduled Searches Setup

- 10 searches are scheduled to run every minute
- 5 longer running searches (~40s)
- 5 shorter running searches (~15s)
- Test configured to run only 3 scheduled concurrently

Scheduled Searches

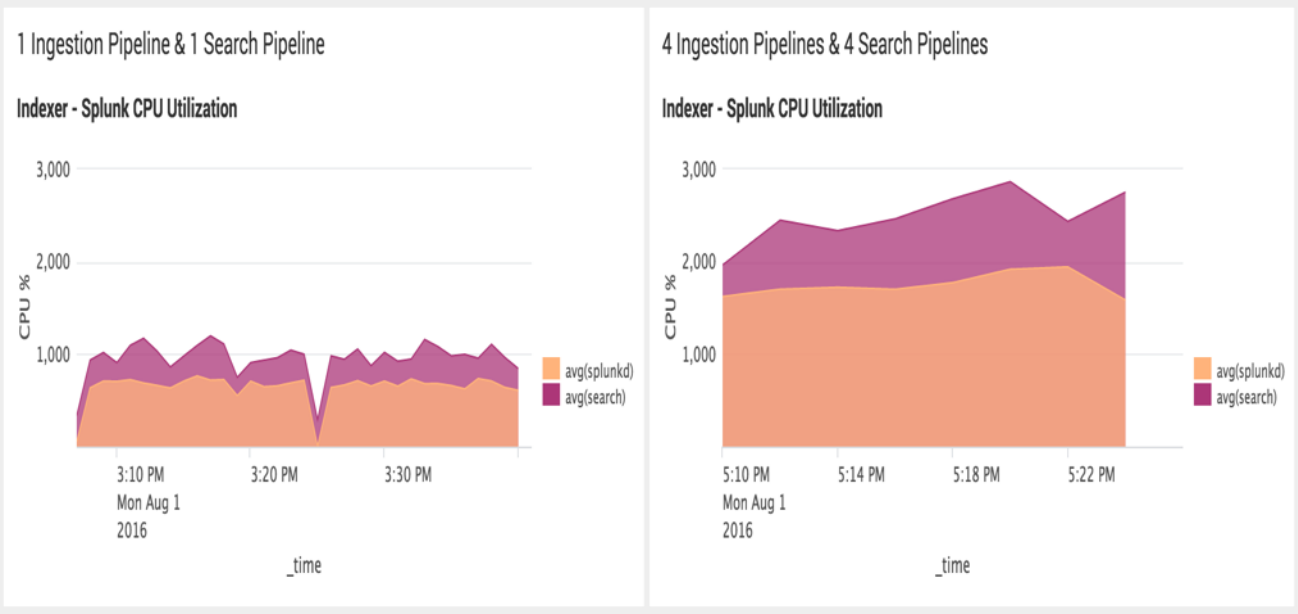




Version	Searches completed
6.2	191
6.5	248

- Skipped vs. Successful Searches – 30 minute window
- 30% Increase in Successful Searches
- This optimization will not utilize additional System Resource

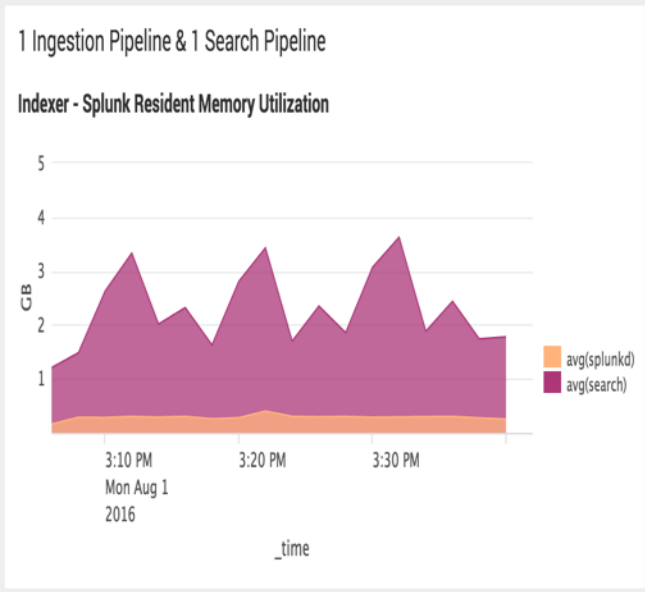
CPU Utilization

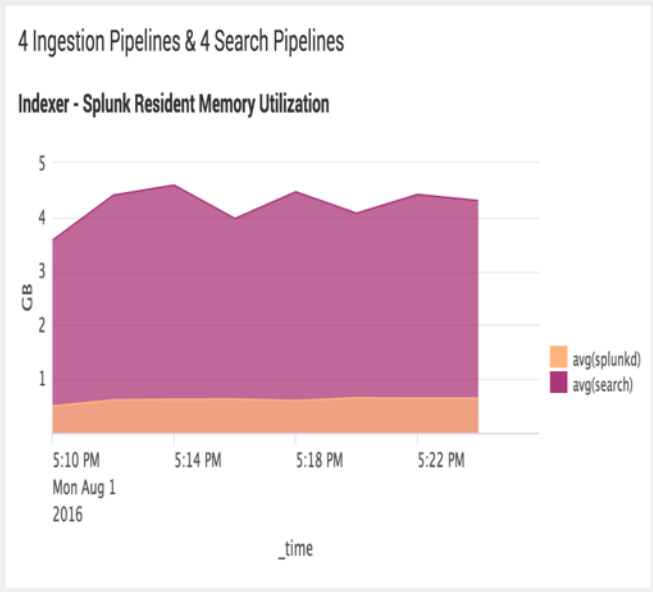


Ingestion Pipelines	Search Pipelines	CPU Utilized
1	1	990 %
4	4	2437 %

- Burst in Indexing Load + Searches
- CPU utilized by splunkd & search process

Memory Utilization

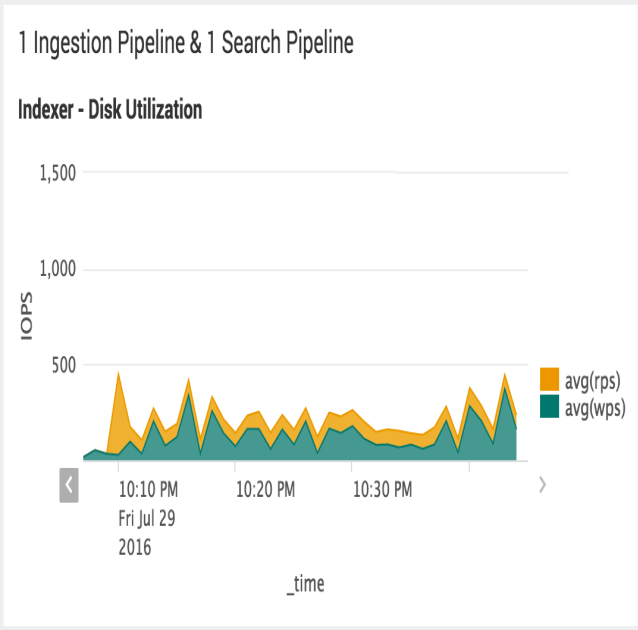


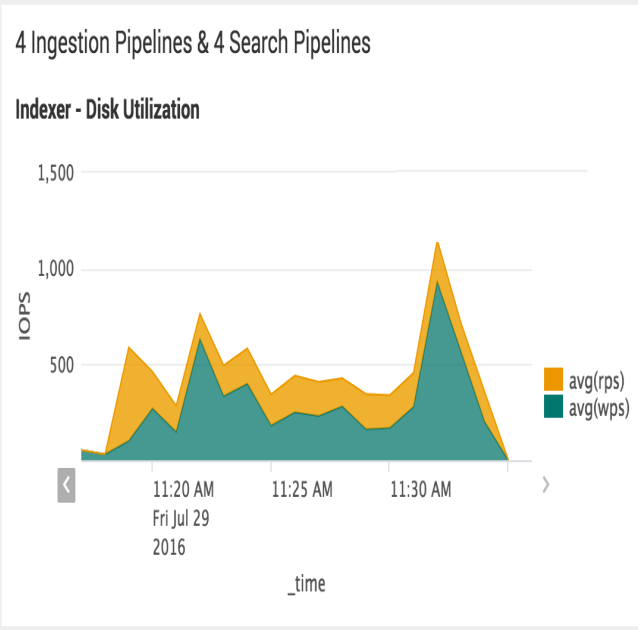


Ingestion Pipelines	Search Pipelines	Memory Utilized
1	1	3.32 GB
4	4	4.59 GB

- Burst in Indexing Load + Searches
- Resident Memory utilized by splunkd & search process

Disk I/O





Ingestion Pipelines	Search Pipelines	Average Disk IOPS
1	1	202
4	4	579

- Burst in Indexing Load + Searches
- Average Read and Writes Operations per second

Final Thoughts

- What is my Current Workload?
  - Data volume – Daily and Peak
  - Search Volume – Concurrent and total
  - System Resource Usage
- How do I approach these features?
  - System significantly under-utilized ?
  - Search Pipelines
    - Lot of Batch mode Searches ?
  - Parallel Ingestion Pipelines
    - Handling Bursts in Data?
    - Reading large number of files in parallel?
- Don’t forget about Horizontal scaling

THANK YOU

splunk>