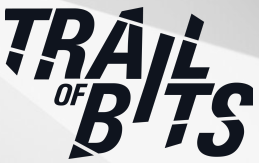


OpenSearch

Benchmark Assessment (Summary Report)

March 3, 2025

Prepared for:

Saurabh Singh | *Amazon*
sisurab@amazon.com

Govind Kamat | *Amazon*
govkamat@amazon.com

Prepared by:

Evan Downing
Riccardo Schirone
Francesco Bertolaccini
Ronald Eytchison

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries and government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence for blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Assessment projects are time-boxed and often rely on information provided by clients, affiliates, or partners. As a result, the findings documented in this report should not be considered a comprehensive list of features in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test software controls and properties. These techniques augment our manual security review work, but each has its limitations: For example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by a project's time and resource constraints.

Table of Contents

About Trail of Bits **1**
Notices and Remarks **2**
Table of Contents **3**
Project Summary **5**
Project Targets **7**
Executive Summary **8**
Engagement Overview 8
Observations and Impact 10
Performance results 10
Performance Trajectory 15
Vectorsearch workload 15
Performance inconsistencies 16
Recommendations 16
Evaluation Setup **18**
Performance Results **19**
Overview 19
Results 20
Performance Overview 20
Sparklines 20
Categorized Results 24
Sort 24
Date histograms 25
Term Aggregations 25
Others 25
Performance Trajectory **28**
Categorized Results 30
Big5 Date Histogram 30
Big5 Range Queries 31
Big5 Sorting 33
Big5 Term Aggregations 35
Big5 Text Querying 37
Vectorsearch **39**
Standard Configuration 39
Sparklines 40
Additional Configuration Experiments 41

Sparklines: Force Merge Enabled 42
Sparklines: Force Merge Disabled 42
Performance Validation 43
Validation of Results 43
Outlier Performance 44
Instance Variation 46
References 48
Appendix 49
Operations 49
Big5 (41 operations) 49
NYC Taxis (7 operations) 51
PMC (6 operations) 52
NOAA (24 operations) 52
Vectorsearch (1 operation) 54
NOAA-Semantic-Search (20 operations) 54
OpenSearch v2.18 - Performance Results 57
Sparklines 57
Categorized Results 59
Sort 59
Date histograms 60
Term Aggregations 60
Others 60
OpenSearch v2.18 - Vectorsearch 62
Standard Configuration 62
Sparklines 63
Additional Configuration Experiments 63
Sparklines: Force Merge Enabled 63
Sparklines: Force Merge Disabled 64
OpenSearch vs. Elasticsearch on Big5 Across Versions 64

Project Summary

Contact Information

The following project manager was associated with this project:

Wendy Hutson, Project Manager
wendy.hutson@trailofbits.com

The following engineering director was associated with this project:

William Woodruff, Engineering Director, Engineering
william.woodruff@trailofbits.com

The following consultants were associated with this project:

Francesco Bertolaccini, Security Engineer II | *Trail of Bits*
francesco.bertolaccini@trailofbits.com

Stefano Bonicatti, Senior Security Engineer | *Trail of Bits*
stefano.bonicatti@trailofbits.com

Henrik Brodin, Principal Security Engineer | *Trail of Bits*
henrik.brodin@trailofbits.com

Dr. Evan Downing, Senior Security Engineer | *Trail of Bits*
evan.downing@trailofbits.com

Ronald Eytchison, Security Engineer I | *Trail of Bits*
ronald.eytchison@trailofbits.com

Alessandro Gario, Senior Security Engineer | *Trail of Bits*
alessandro.gario@trailofbits.com

Riccardo Schirone, Security Engineer II | *Trail of Bits*
riccardo.schirone@trailofbits.com

Brad Swain, Security Engineer II | *Trail of Bits*
brad.swain@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
August 27, 2024	Kickoff meeting with client
September – December	Weekly status updates
December 20, 2024	Delivery of report draft
December 20–27, 2024	Review period for feedback
January - February 2025	Continued iterative feedback
March 3, 2025	Delivery of final summary report

Project Targets

The engagement involved executing OpenSearch Benchmark workloads on both OpenSearch and Elasticsearch.

OpenSearch Benchmark

Repository	https://github.com/opensearch-project/opensearch-benchmark
Type	Python
Platform	GNU/Linux

OpenSearch

Repository	https://github.com/opensearch-project/OpenSearch
Type	Java
Platform	GNU/Linux

Elasticsearch

Repository	https://github.com/elastic/elasticsearch
Type	Java
Platform	GNU/Linux

Executive Summary

Engagement Overview

Amazon Corporate LLC (Amazon) engaged Trail of Bits to conduct a benchmark comparison of two prominent search-and-analysis software tools: OpenSearch and Elasticsearch. Both products were stress tested using OpenSearch Benchmark (OSB) and its associated workloads to evaluate their query performance.

Eight engineering consultants conducted the review from August 2 to December 31, 2024. Trail of Bits developed code that automates repeatable testing and analysis of OpenSearch and Elasticsearch on Amazon Web Services (AWS) cloud infrastructure. Trail of Bits was not involved in modifying the OpenSearch codebase with performance improvements identified during benchmarking.

The goal of this review is to compare the service time values of OpenSearch and Elasticsearch on OpenSearch Benchmark workloads. Service time (measured in milliseconds) represents how long it takes for a request (i.e., query) to receive a response [4]. This includes overhead (network latency, load balancer overhead, serialization/deserialization, etc.). OpenSearch Benchmark records the 90th percentile (p90) of service times executed for each operation. These 90th percentile values are analyzed throughout this report.

The scope was limited to the Apache v2 (OpenSearch) and Elastic 2.0 (Elasticsearch) licensed versions of the engines, and did not include proprietary systems. The results can be used to direct future development of individual components for each engine.

Six OSB workloads were evaluated, testing key capabilities:

1. Big5: Performs text querying, sorting, date histogram, range queries, and terms aggregation
2. NYC Taxis: Queries a dataset of yellow taxi rides taken in New York City (NYC) during 2015. This workload exercises a variety of search queries.
3. PMC: Queries a subset of PubMed Central (PMC) articles. This workload primarily performs text queries.
4. NOAA: Queries daily weather measurements collected by the National Oceanic and Atmospheric Administration (NOAA) using various aggregation queries.
5. Vectorsearch: Queries a dataset of one billion vectors using approximate K-Nearest Neighbor (KNN) search
6. NOAA Semantic Search: Performs semantic search queries on the NOAA dataset.

a. **NOTE**: Because Elasticsearch does not have an equivalent feature for hybrid searches, we do not compare it to OpenSearch for this workload. Instead, we compare only different OpenSearch versions.

Observations and Impact

All measurements in this report represent 90th percentile service time values. During the engagement, we made several observations related to performance results and trajectory, the vectorsearch workload, and performance inconsistencies, which are described below.

We calculated the median of each workload operation's p90 service time. When grouping operations (for example, to measure performance across workload categories or the entire workload) we calculated the geometric mean (geomean) of these medians. We use geometric mean because we are averaging across different operations (ops).

Performance results

We calculated the geomean of median p90 service times across each workload:

- OpenSearch is faster by 1.56x on Big5 compared to Elasticsearch
- OpenSearch is faster by 7.82x on NYC Taxis compared to Elasticsearch
- OpenSearch is slower by 1.12x on PMC compared to Elasticsearch
- OpenSearch is slower by 1.82x on NOAA compared to Elasticsearch

For Big5 in particular, comparing the queries according to their categories represented in that workload:

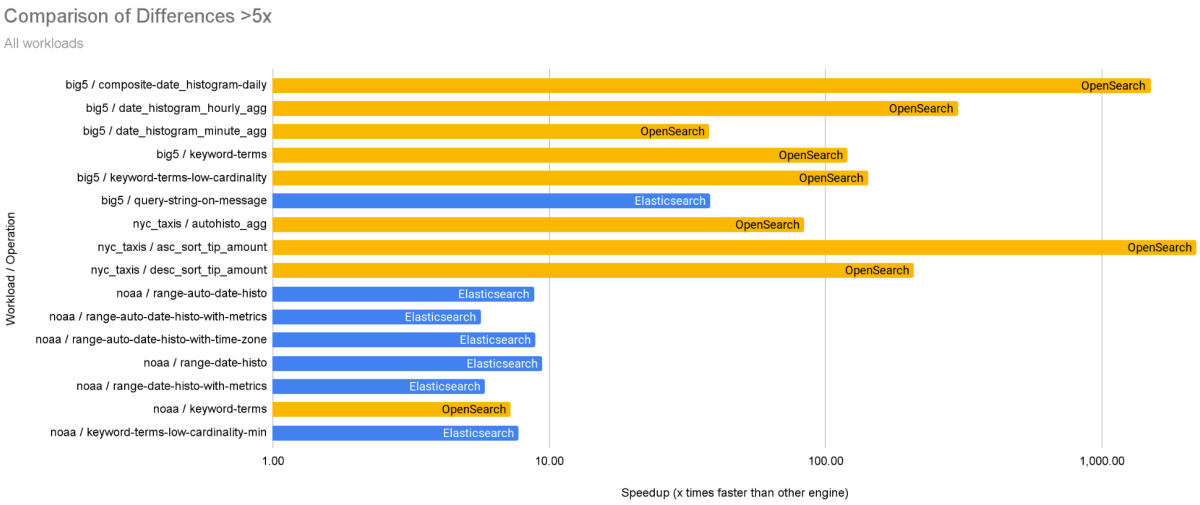
- Date Histogram: OpenSearch is faster by 16.55x
- Range queries: OpenSearch is faster by 1.02x
- Sorting: OpenSearch is faster by 1.05x
- Term Aggregations: OpenSearch is faster by 3.38x
- Text queries: OpenSearch is slower by 2.42x

Geomean of Median p90 Service Time (ms)	OpenSearch v2.17.1	Elasticsearch v8.15.4	OpenSearch is slower/faster than Elasticsearch
Big5 (all ops aggregated)	12.09	18.83	1.56x faster
Date Histogram	124.79	2,064.61	16.55x faster
Range Queries	1.47	1.49	1.02x faster
Sorting	5.82	6.14	1.05x faster
Term Aggregations	104.90	354.52	3.38x faster
Text Querying	18.11	7.47	2.42x slower
NYC Taxis (all ops aggregated)	24.67	193.01	7.82x faster

PMC (all ops aggregated)	6.85	6.09	1.12x slower
NOAA (all ops aggregated)	979.65	538.91	1.82x slower

Additional considerations:

- When considering operations with differences greater than 5x in median p90 service time between Elasticsearch and OpenSearch (21% of all operations — 16 out of 78), OpenSearch is several orders of magnitude faster (7x-2,200x) than Elasticsearch in 56% of these operations (9 out of 16). In contrast, Elasticsearch is faster by a smaller degree (6x-38x) for 44% of those operations (7 out of 16). See the graph below.
- For operations with differences less than 5x between the two engines, Elasticsearch is faster on roughly 70% of operations compared to OpenSearch.
- When considering the median p90 service time values for *all* operations across *all* workloads, Elasticsearch is faster on 64% of operations compared to OpenSearch.



The following tables show differences in service times for all operations in all workloads grouped by query category. Speedup represents how many times faster one engine is compared to the other, given median p90 service times.

Category	Workload / Operation	x Speedup	Faster
Aggregation	nyc_taxis / autohisto_agg	83.76	OpenSearch
Aggregation	nyc_taxis / date_histogram_agg	1.15	OpenSearch
Aggregation	nyc_taxis / distance_amount_agg	4.18	Elasticsearch
Aggregation	pmc / articles_monthly_agg_cached	1.37	Elasticsearch

Aggregation	pmc / articles_monthly_agg_uncached	1.12	OpenSearch

Category	Workload / Operation	x Speedup	Faster
Date Histogram	big5 / composite-date_histogram-daily	1,494.42	OpenSearch
Date Histogram	big5 / date_histogram_hourly_agg	301.17	OpenSearch
Date Histogram	big5 / date_histogram_minute_agg	38.02	OpenSearch
Date Histogram	big5 / range-auto-date-histo	3.74	Elasticsearch
Date Histogram	big5 / range-auto-date-histo-with-metrics	3.69	Elasticsearch
Date Histogram	noaa / date-histo-entire-range	1.04	OpenSearch
Date Histogram	noaa / date-histo-geohash-grid	4.16	Elasticsearch
Date Histogram	noaa / date-histo-geotile-grid	4.88	Elasticsearch
Date Histogram	noaa / date-histo-histo	4.08	Elasticsearch
Date Histogram	noaa / date-histo-numeric-terms	1.04	Elasticsearch
Date Histogram	noaa / date-histo-string-significant-terms-via-default-strategy	1.02	Elasticsearch
Date Histogram	noaa / date-histo-string-significant-terms-via-global-ords	1.02	Elasticsearch
Date Histogram	noaa / date-histo-string-significant-terms-via-map	1.04	Elasticsearch
Date Histogram	noaa / date-histo-string-terms-via-default-strategy	1.04	Elasticsearch
Date Histogram	noaa / date-histo-string-terms-via-global-ords	1.03	Elasticsearch
Date Histogram	noaa / date-histo-string-terms-via-map	1.04	Elasticsearch

Category	Workload / Operation	x Speedup	Faster

Range & Date Histogram	noaa / range-auto-date-histo	8.84	Elasticsearch
Range & Date Histogram	noaa / range-auto-date-histo-with-metrics	5.66	Elasticsearch
Range & Date Histogram	noaa / range-date-histo	9.47	Elasticsearch
Range & Date Histogram	noaa / range-date-histo-with-metrics	5.86	Elasticsearch

Category	Workload / Operation	x Speedup	Faster
Range Queries	big5 / range	1.06	OpenSearch
Range Queries	big5 / keyword-in-range	1.05	OpenSearch
Range Queries	big5 / range_field_conjunction_big_range_big_term_query	1.06	Elasticsearch
Range Queries	big5 / range_field_conjunction_small_range_big_term_query	1.11	Elasticsearch
Range Queries	big5 / range_field_conjunction_small_range_small_term_query	1.06	Elasticsearch
Range Queries	big5 / range_field_disjunction_big_range_small_term_query	1.05	Elasticsearch
Range Queries	big5 / range-agg-1	1.28	OpenSearch
Range Queries	big5 / range-agg-2	1.17	OpenSearch
Range Queries	big5 / range-numeric	1.08	Elasticsearch
Range Queries	nyc_taxis / range	3.10	Elasticsearch
Range Queries	noaa / range-aggregation	1.25	Elasticsearch
Range Queries	noaa / range-numeric-significant-terms	1.88	Elasticsearch

Category	Workload / Operation	x Speedup	Faster
Sorting	big5 / asc_sort_timestamp	4.66	OpenSearch
Sorting	big5 / asc_sort_timestamp_can_match_shortcut	3.77	OpenSearch
Sorting	big5 /	3.76	OpenSearch

	asc_sort_timestamp_no_can_match_shortcut		
Sorting	big5 / asc_sort_with_after_timestamp	2.15	OpenSearch
Sorting	big5 / desc_sort_timestamp	1.48	Elasticsearch
Sorting	big5 / desc_sort_timestamp_can_match_shortcut	1.36	Elasticsearch
Sorting	big5 / desc_sort_timestamp_no_can_match_shortcut	1.37	Elasticsearch
Sorting	big5 / sort_keyword_can_match_shortcut	1.01	OpenSearch
Sorting	big5 / desc_sort_with_after_timestamp	2.76	Elasticsearch
Sorting	big5 / sort_keyword_no_can_match_shortcut	1.01	OpenSearch
Sorting	big5 / sort_numeric_asc	2.81	Elasticsearch
Sorting	big5 / sort_numeric_asc_with_match	1.12	Elasticsearch
Sorting	big5 / sort_numeric_desc	2.58	Elasticsearch
Sorting	big5 / sort_numeric_desc_with_match	1.13	Elasticsearch
Sorting	nyc_taxis / asc_sort_tip_amount	2,195.56	OpenSearch
Sorting	nyc_taxis / desc_sort_tip_amount	208.96	OpenSearch

Category	Workload / Operation	x Speedup	Faster
Term Aggregations	big5 / cardinality-agg-high	3.14	Elasticsearch
Term Aggregations	big5 / cardinality-agg-low	1.79	Elasticsearch
Term Aggregations	big5 / composite_terms-keyword	1.06	Elasticsearch
Term Aggregations	big5 / composite-terms	1.03	Elasticsearch
Term Aggregations	big5 / keyword-terms	120.44	OpenSearch
Term Aggregations	big5 / keyword-terms-low-cardinality	142.53	OpenSearch
Term Aggregations	big5 / multi_terms-keyword	1.78	OpenSearch
Term Aggregations	noaa / keyword-terms	7.26	OpenSearch
Term Aggregations	noaa / keyword-terms-low-cardinality	3.62	OpenSearch
Term Aggregations	noaa / keyword-terms-low-cardinality-min	7.78	Elasticsearch

Term Aggregations	noaa / keyword-terms-min	1.02	OpenSearch
Term Aggregations	noaa / keyword-terms-numeric-terms	1.04	OpenSearch
Term Aggregations	noaa / numeric-terms-numeric-terms	1.04	Elasticsearch

Category	Workload / Operation	x Speedup	Faster
Text Querying	big5 / default	1.79	Elasticsearch
Text Querying	big5 / query-string-on-message	38.36	Elasticsearch
Text Querying	big5 / query-string-on-message-filtered	2.69	Elasticsearch
Text Querying	big5 / query-string-on-message-filtered-sorted-num	1.04	OpenSearch
Text Querying	big5 / scroll	1.11	Elasticsearch
Text Querying	big5 / term	1.04	Elasticsearch
Text Querying	nyc_taxis / default	1.90	Elasticsearch
Text Querying	pmc / default	1.01	Elasticsearch
Text Querying	pmc / phrase	1.07	OpenSearch
Text Querying	pmc / scroll	1.63	Elasticsearch
Text Querying	pmc / term	1.08	Elasticsearch

Performance Trajectory

OpenSearch 2.17.1 demonstrated mixed results. Certain operations degraded in performance for Big5, NOAA, and NOAA Semantic Search workloads, while other operations improved in performance for NYC Taxis and PMC workloads.

Elasticsearch 8.15.4 also showed mixed results across workloads, with performance improvements in all four datasets compared to version 8.13.4. Most notably, it achieved a 22% speed increase for Big5 workloads and a 10% improvement for NYC taxi data.

Vectorsearch workload

In our standard configuration, OpenSearch (NMSLIB engine) and OpenSearch (FAISS engine) are faster than Elasticsearch (Lucene engine).

In our standard configuration, OpenSearch (Lucene engine) is slower than Elasticsearch (Lucene engine). However, in additional experiments, we found that this gap can be narrowed with a lower JVM Heap limit and modified snapshot process.

Performance inconsistencies

During our evaluation, we observed non-statistical outliers in engine performance, particularly for individual operations—at times, both engines were unusually slow. While we made efforts to identify and mitigate these scenarios, we could not understand the root cause of these anomalous runs. Though the overall conclusions above did not change, these scenarios should be noted and explored further. All outliers were still included in the results, as there was no systematic way of removing them.

Recommendations

From our observations running OpenSearch and Elasticsearch dozens of times over the course of our engagement, we recommend the following:

- OpenSearch should focus on improving the performance of the following operations.
    - Descending sort operations (particularly for timestamps) for the Big5 workload.
    - Query string on message operations for the Big5 workload.
    - Range auto date histogram operations for the Big5 workload.
    - Most operations in the NOAA workload (particularly Range & Date Histogram operations).
- Performing benchmark experiments
    - Always run tests (a set of five runs of each workload) on newly created instances. We find that results may not be realistic if workloads are only run on the same exact instance. If workloads are not run on new instances, variations in workload performance may not be observed, which would skew a user's expectations.
    - After collecting data, measure the *p-value* and statistical power to ensure that results are statistically reliable: measuring the p-value across different runs with the same configuration can be helpful in detecting anomalies (i.e., the p-value when comparing "similar" runs is expected to be high); measuring against a different configuration (e.g., different setup, different engine) is necessary to assess that the change produces statistically different results.
    - Collect all logs on the target instance and enable collecting low-level system information during runs. This will help debug performance inconsistencies for both engines.
    - The configuration used during benchmarking should be as close as possible to the out-of-the-box experience. If any changes are necessary for a fair

benchmark, they should be clearly documented, and the reason why they would not be good for a default configuration should be mentioned.

- A snapshot approach that may create more consistent results is to flush the index, refresh the index, and then wait for merges to complete before taking a snapshot. We found initial promising results in testing vectorsearch, but have not extensively tested this strategy.

Evaluation Setup

We tested OpenSearch and Elasticsearch with similar configurations to remain objective in our findings. As we adjusted to OpenSearch (with guidance from Amazon), we made similar adjustments to Elasticsearch (based on documentation). Each workload was executed using OSB five times consecutively to collect 90th percentile `service_time` values. The first run was always discarded as a warmup run. This was done to ensure most of the data from the workload corpus was loaded into memory and caches were properly primed. The remaining runs were used to calculate the results of this report.

Testing infrastructure was provisioned on AWS using Terraform for consistent and reproducible deployments. Tooling is available here [5] for reproducibility. For most workloads, we used `c5d.2xlarge` instances (x86 architecture) for both the load generation host and the target cluster. The vectorsearch workload required a different setup, using a three-node cluster of `r6gd.4xlarge` instances (ARM architecture) for the target cluster and a `c5d.4xlarge` instance for load generation. The vectorsearch workload uses larger `4xlarge` instances to have more memory and uses a multi-node setup to have replicas. Finally, we used SSD instance-based storage for all setups.

Each test scenario was isolated and run on dedicated load-generation hosts and target clusters. The load-generation host performed multiple functions: data indexing, snapshot management (creation and restoration), and benchmark execution using OSB. The target cluster ran either OpenSearch or Elasticsearch, configured according to each test's requirements.

All instances were standard AWS instances without any special optimizations. OSB reported that no errors occurred during query execution of the tests (i.e. the error rate was 0%). However, note that OSB currently does not validate the correctness of query results, apart from keeping track of returned error status from the API calls. We performed manual spot-checks of random query results across different workloads to ensure result validity. Furthermore, we verified that our Big5 results were similar to OpenSearch's internal benchmark results [6].

In order to compare the vectorsearch workload, OpenSearch and Elasticsearch had to be configured differently. The OpenSearch query and index were modified for Elasticsearch to conform to the Elasticsearch APIs. Additionally, the Elasticsearch query used a different ID field instead of the default `_id` field used by OpenSearch. Our nightly results for the standard configuration also include some results before two minor configuration changes were made on December 09, 2024. The first change was updating OpenSearch to use the default `_id` field. In earlier runs, it used a different ID field like Elasticsearch. The second change was enforcing the exact document counts in our index. In earlier runs, we allowed a small number of duplicate documents (up to 0.05% of the corpus size). We included the earlier results because we believe the impact of these changes is negligible.

Performance Results

Overview

In this section, we highlight performance measurements for OpenSearch v2.17.1 and Elasticsearch v8.15.4. The following results compare the workloads: Big5, NYC Taxis, PMC, and NOAA.

The latest version at the time of our testing was OpenSearch v2.18.0. However, the Lucene version used by OpenSearch was upgraded in v2.18.0 from Lucene 9.11.1 to Lucene 9.12.0. This resulted in the performance of a few queries being impacted as described in a Github issue [7]. Since the issue is within the Lucene codebase, rather than OpenSearch, comparisons with Elasticsearch are provided for v2.17.1, while v2.18.0 results are added to the Appendix.

We tested each engine every day at the same time on the same instance type (from December 1, 2024 – December 16, 2024) on the four OSB workloads. We collected between 13 and 15 tests per workload per engine. Recall that each workload was executed four times for each test: there were 5 runs, with the first (warmup) run discarded. The number of times each operation was executed during each run ranged from hundreds to thousands (depending on OSB's configuration – the `test-iterations` parameter). This resulted in thousands to tens of thousands of sample measurements per operation. We believe this is a large enough sample size to draw reliable conclusions.

We calculated the median of each workload operation's p90 service time. We did this because we observed non-trivial variations in performance in several runs for both OpenSearch and Elasticsearch. These outliers can impact the arithmetic average. We chose not to statistically exclude these outliers (e.g., using standard deviation or quartiles as the exclusion criteria) because the results do not necessarily follow a Gaussian (normal) distribution. Therefore, we believe the median across this large number of independent data points is most representative of the summary statistics we calculated for OpenSearch and Elasticsearch.

We report our benchmark findings of vectorsearch separately because of the distinct nature of this workload and because additional iterative configurations were required to achieve results similar to OpenSearch's own internal benchmarking results. We report NOAA Semantic Search results in the Performance Trajectory section below, as those results can only be compared between OpenSearch versions (Elasticsearch does not support running the workload at this time).

Results

Performance Overview

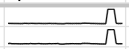
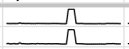
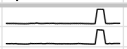
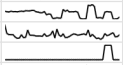
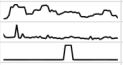
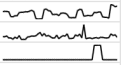
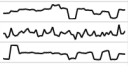
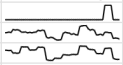
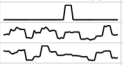
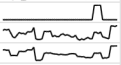
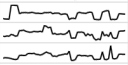
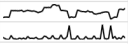
To measure statistical certainty, we calculated the p-value and statistical power of each operation in each workload for each engine version. To do this, we looked at *all* reported values for each execution of each run (instead of just the 90th percentile value determined by OSB) to ensure a holistic view of statistical significance. For p-values less than 0.05 (indicating a statistically significant difference in performance), OpenSearch outperformed Elasticsearch on 28 of 78 tasks (35.9%) using geometric mean. Elasticsearch outperformed OpenSearch on 50 of 78 tasks (64.1%). We discuss statistical certainty more in the Performance Validation section below.

Notably, while OpenSearch is slower on most tasks, the tasks it is faster on are 1–3 orders of magnitude faster than Elasticsearch.

Sparklines

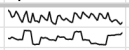
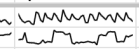
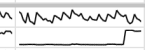
For transparency we created sparklines [3] visualizing *all* of the data (plotting the 90th percentile of the service times reported by OSB on the y-axis, sorted by timestamp of when that benchmark was executed on the x-axis) for each operation in each workload. While aggregations communicate a single number to compare (e.g., median), they can also paint an incomplete picture for readers. In this case, sparklines communicate consistency and variation across executions of each operation in each workload, as well as the prevalence of outliers. Below, we compare different versions of OpenSearch to the latest version of Elasticsearch for five workloads.

This first set of charts does not use the same scale and instead shows variations in 90th percentile service time values. Note that a smaller value is better (i.e., faster). These charts communicate *consistency in performance*. For example, the `asc_sort_timestamp` operation in the Big5 workload has consistent performance in OpenSearch, save an outlier. Initially it may seem like Elasticsearch has inconsistencies in performance for that operation, but because the y-axis is different the viewer cannot tell that it ranges from 22-23 milliseconds.

Workload	Category	Operation	OpenSearch 2.16.0	OpenSearch 2.17.1	OpenSearch 2.18.0	Elasticsearch 8.15.4
big5	Sorting	asc_sort_timestamp				
big5	Sorting	asc_sort_timestamp_can_match_shortcut				
big5	Sorting	asc_sort_timestamp_no_can_match_shortcut				
big5	Sorting	asc_sort_with_after_timestamp				
big5	Term Aggregations	cardinality-agg-high				
big5	Term Aggregations	cardinality-agg-low				
big5	Date Histogram	composite-date_histogram-daily				
big5	Term Aggregations	composite-terms				
big5	Term Aggregations	composite_terms-keyword				
big5	Date Histogram	date_histogram_hourly_agg				
big5	Date Histogram	date_histogram_minute_agg				
big5	General Operations	default				
big5	Sorting	desc_sort_timestamp				
big5	Sorting	desc_sort_timestamp_can_match_shortcut				
big5	Sorting	desc_sort_timestamp_no_can_match_shortcut				
big5	Sorting	desc_sort_with_after_timestamp				
big5	Range Queries	keyword-in-range				
big5	Term Aggregations	keyword-terms				
big5	Term Aggregations	keyword-terms-low-cardinality				
big5	Term Aggregations	multi_terms-keyword				
big5	Text Querying	query-string-on-message				
big5	Text Querying	query-string-on-message-filtered				
big5	Text Querying	query-string-on-message-filtered-sorted-num				
big5	Range Queries	range				
big5	Range Queries	range-agg-1				
big5	Range Queries	range-agg-2				
big5	Date Histogram	range-auto-date-histo				
big5	Date Histogram	range-auto-date-histo-with-metrics				
big5	Range Queries	range-numeric				
big5	Range Queries	range_field_conjunction_big_range_big_term_query				
big5	Range Queries	range_field_conjunction_small_range_big_term_query				
big5	Range Queries	range_field_conjunction_small_range_small_term_query				
big5	Range Queries	range_field_disjunction_big_range_small_term_query				
big5	General Operations	scroll				
big5	Sorting	sort_keyword_can_match_shortcut				
big5	Sorting	sort_keyword_no_can_match_shortcut				
big5	Sorting	sort_numeric_asc				
big5	Sorting	sort_numeric_asc_with_match				
big5	Sorting	sort_numeric_desc				
big5	Sorting	sort_numeric_desc_with_match				
big5	Text Querying	term				

Workload	Category	Operation	OpenSearch 2.16.0	OpenSearch 2.17.1	OpenSearch 2.18.0	Elasticsearch 8.15.4
nyc_taxis	Sorting	asc_sort_tip_amount				
nyc_taxis	Aggregation	autohisto_agg				
nyc_taxis	Aggregation	date_histogram_agg				
nyc_taxis	General Operations	default				
nyc_taxis	Sorting	desc_sort_tip_amount				
nyc_taxis	Aggregation	distance_amount_agg				
nyc_taxis	Range Queries	range				

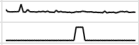
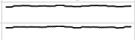
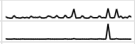
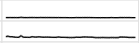
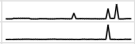
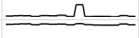
Workload	Category	Operation	OpenSearch 2.16.0	OpenSearch 2.17.1	OpenSearch 2.18.0	Elasticsearch 8.15.4
pmc	Aggregation	articles_monthly_agg_cached				
pmc	Aggregation	articles_monthly_agg_uncached				
pmc	General Operations	default				
pmc	Text Querying	phrase				
pmc	General Operations	scroll				
pmc	Text Querying	term				

Workload	Category	Operation	OpenSearch 2.16.0	OpenSearch 2.17.1	OpenSearch 2.18.0	Elasticsearch 8.15.4
noaa	Date Histogram	date-histo-entire-range				
noaa	Date Histogram	date-histo-geohash-grid				
noaa	Date Histogram	date-histo-geotile-grid				
noaa	Date Histogram	date-histo-histo				
noaa	Date Histogram	date-histo-numeric-terms				
noaa	Date Histogram	date-histo-string-significant-terms-via-default-strategy				
noaa	Date Histogram	date-histo-string-significant-terms-via-global-ords				
noaa	Date Histogram	date-histo-string-significant-terms-via-map				
noaa	Date Histogram	date-histo-string-terms-via-default-strategy				
noaa	Date Histogram	date-histo-string-terms-via-global-ords				
noaa	Date Histogram	date-histo-string-terms-via-map				
noaa	Term Aggregations	keyword-terms				
noaa	Term Aggregations	keyword-terms-low-cardinality				
noaa	Term Aggregations	keyword-terms-low-cardinality-min				
noaa	Term Aggregations	keyword-terms-min				
noaa	Term Aggregations	keyword-terms-numeric-terms				
noaa	Term Aggregations	numeric-terms-numeric-terms				
noaa	Range Queries	range-aggregation				
noaa	Range & Date Histogram	range-auto-date-histo				
noaa	Range & Date Histogram	range-auto-date-histo-with-metrics				
noaa	Range & Date Histogram	range-auto-date-histo-with-time-zone				
noaa	Range & Date Histogram	range-date-histo				
noaa	Range & Date Histogram	range-date-histo-with-metrics				
noaa	Range Queries	range-numeric-significant-terms				

Workload	Category	Operation	OpenSearch 2.16.0	OpenSearch 2.17.1	OpenSearch 2.18.0
noaa_semantic_search	Aggregation	aggs-query-date-histo-geohash-grid-hybrid			
noaa_semantic_search	Aggregation	aggs-query-date-histo-geohash-grid-hybrid-one-subquery			
noaa_semantic_search	Aggregation	aggs-query-date-histo-geohash-grid-hybrid-one-subquery-large-subset			
noaa_semantic_search	Aggregation	aggs-query-date-histo-geohash-grid-hybrid-one-subquery-medium-subset			
noaa_semantic_search	Aggregation	aggs-query-min-avg-sum-hybrid			
noaa_semantic_search	Aggregation	aggs-query-min-avg-sum-hybrid-one-subquery			
noaa_semantic_search	Aggregation	aggs-query-min-avg-sum-hybrid-one-subquery-large-subset			
noaa_semantic_search	Aggregation	aggs-query-min-avg-sum-hybrid-one-subquery-medium-subset			
noaa_semantic_search	Aggregation	aggs-query-range-numeric-significant-terms-hybrid			
noaa_semantic_search	Aggregation	aggs-query-range-numeric-significant-terms-hybrid-one-subquery			
noaa_semantic_search	Aggregation	aggs-query-range-numeric-significant-terms-hybrid-one-subquery-large-subset			
noaa_semantic_search	Aggregation	aggs-query-range-numeric-significant-terms-hybrid-one-subquery-medium-subset			
noaa_semantic_search	Aggregation	aggs-query-term-min-hybrid			
noaa_semantic_search	Aggregation	aggs-query-term-min-hybrid-one-subquery			
noaa_semantic_search	Aggregation	aggs-query-term-min-hybrid-one-subquery-large-subset			
noaa_semantic_search	Aggregation	aggs-query-term-min-hybrid-one-subquery-medium-subset			
noaa_semantic_search	Hybrid Query	hybrid-query-only-range			
noaa_semantic_search	Hybrid Query	hybrid-query-only-range-large-subset			
noaa_semantic_search	Hybrid Query	hybrid-query-only-range-medium-subset			
noaa_semantic_search	Hybrid Query	hybrid-query-only-term-range-date			

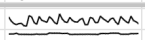
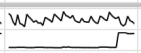
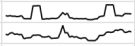
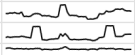
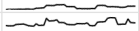
Next, we zoom in on the recent versions of OpenSearch (v2.17.1) and Elasticsearch (v8.15.4). We plot sparklines using the *same* scale for each operation (noting the maximum and minimum value for each plot) to illustrate how service times compare between each engine. Note that a smaller value is better (i.e., faster). We do not compare NOAA Semantic Search because Elasticsearch does not currently support that workload.

These charts communicate a *comparison of performance*. For example, observe the `asc_sort_timestamp` operation for the Big5 workload again. In the chart below, Elasticsearch's service time values are all *higher* than OpenSearch's. This shows the reader that (for this particular operation) in the worst case outlier OpenSearch will be roughly as fast as Elasticsearch, but in most cases it is *faster*.

Workload	Category	Operation	OpenSearch 2.17.1	Elasticsearch 8.15.4	Min	Max
big5	Sorting	asc_sort_timestamp			4.66	23.88
big5	Sorting	asc_sort_timestamp_can_match_shortcut			4.72	20.37
big5	Sorting	asc_sort_timestamp_no_can_match_shortcut			4.72	20.21
big5	Sorting	asc_sort_with_after_timestamp			148.19	469.55
big5	Term Aggregations	cardinality-agg-high			702.33	2,578.80
big5	Term Aggregations	cardinality-agg-low			0.89	2.63
big5	Date Histogram	composite-date_histogram-daily			1.04	1,813.50
big5	Term Aggregations	composite-terms			214.22	248.16
big5	Term Aggregations	composite_terms-keyword			309.13	386.47
big5	Date Histogram	date_histogram_hourly_agg			4.49	1,526.28
big5	Date Histogram	date_histogram_minute_agg			31.04	1,333.46
big5	General Operations	default			1.18	3.02
big5	Sorting	desc_sort_timestamp			3.05	47.62
big5	Sorting	desc_sort_timestamp_can_match_shortcut			2.61	11.13
big5	Sorting	desc_sort_timestamp_no_can_match_shortcut			2.60	11.08
big5	Sorting	desc_sort_with_after_timestamp			64.79	346.69
big5	Range Queries	keyword-in-range			10.77	61.03
big5	Term Aggregations	keyword-terms			23.35	3,359.73
big5	Term Aggregations	keyword-terms-low-cardinality			19.02	3,315.97
big5	Term Aggregations	multi_terms-keyword			729.30	1,394.02
big5	Text Querying	query-string-on-message			3.29	145.69
big5	Text Querying	query-string-on-message-filtered			9.25	43.48
big5	Text Querying	query-string-on-message-filtered-sorted-num			22.36	42.82
big5	Range Queries	range			6.95	9.95
big5	Range Queries	range-agg-1			0.90	1.60
big5	Range Queries	range-agg-2			0.89	1.66
big5	Date Histogram	range-auto-date-histo			2,149.37	9,237.39
big5	Date Histogram	range-auto-date-histo-with-metrics			4,978.15	20,694.60
big5	Range Queries	range-numeric			0.64	1.08
big5	Range Queries	range_field_conjunction_big_range_big_term_query			0.75	1.33
big5	Range Queries	range_field_conjunction_small_range_big_term_query			0.65	1.10
big5	Range Queries	range_field_conjunction_small_range_small_term_query			0.73	1.32
big5	Range Queries	range_field_disjunction_big_range_small_term_query			0.74	1.22
big5	General Operations	scroll			204.15	370.59
big5	Sorting	sort_keyword_can_match_shortcut			2.02	2.91
big5	Sorting	sort_keyword_no_can_match_shortcut			2.04	2.93
big5	Sorting	sort_numeric_asc			2.50	10.32
big5	Sorting	sort_numeric_asc_with_match			0.67	1.25
big5	Sorting	sort_numeric_desc			3.10	9.75
big5	Sorting	sort_numeric_desc_with_match			0.65	1.24
big5	Text Querying	term			0.75	1.23

Workload	Category	Operation	OpenSearch 2.17.1	Elasticsearch 8.15.4	Min	Max
nyc_taxis	Sorting	asc_sort_tip_amount			2.56	6,515.93
nyc_taxis	Aggregation	autohisto_agg			2.89	304.68
nyc_taxis	Aggregation	date_histogram_agg			2.84	5.25
nyc_taxis	General Operations	default			1.13	3.43
nyc_taxis	Sorting	desc_sort_tip_amount			27.53	6,465.69
nyc_taxis	Aggregation	distance_amount_agg			2,500.91	13,165.49
nyc_taxis	Range Queries	range			55.50	199.40

Workload	Category	Operation	OpenSearch 2.17.1	Elasticsearch 8.15.4
pmc	Aggregation	articles_monthly_agg_cached		
pmc	Aggregation	articles_monthly_agg_uncached		
pmc	General Operations	default		
pmc	Text Querying	phrase		
pmc	General Operations	scroll		
pmc	Text Querying	term		

Workload	Category	Operation	OpenSearch 2.17.1	Elasticsearch 8.15.4	Min	Max
noaa	Date Histogram	date-histo-entire-range			1.50	2.53
noaa	Date Histogram	date-histo-geohash-grid			306.83	1,497.47
noaa	Date Histogram	date-histo-geotile-grid			347.98	1,931.89
noaa	Date Histogram	date-histo-histo			379.72	1,804.28
noaa	Date Histogram	date-histo-numeric-terms			1,936.61	3,009.48
noaa	Date Histogram	date-histo-string-significant-terms-via-default-strategy			2,522.88	3,002.71
noaa	Date Histogram	date-histo-string-significant-terms-via-global-ords			2,514.45	3,046.83
noaa	Date Histogram	date-histo-string-significant-terms-via-map			9,719.37	13,016.30
noaa	Date Histogram	date-histo-string-terms-via-default-strategy			2,220.74	2,714.16
noaa	Date Histogram	date-histo-string-terms-via-global-ords			2,202.69	2,724.13
noaa	Date Histogram	date-histo-string-terms-via-map			9,340.31	12,636.05
noaa	Term Aggregations	keyword-terms			141.17	1,245.52
noaa	Term Aggregations	keyword-terms-low-cardinality			2.47	12.53
noaa	Term Aggregations	keyword-terms-low-cardinality-min			188.78	1,649.12
noaa	Term Aggregations	keyword-terms-min			1,924.97	2,379.79
noaa	Term Aggregations	keyword-terms-numeric-terms			5,132.62	7,055.46
noaa	Term Aggregations	numeric-terms-numeric-terms			1,657.14	2,043.05
noaa	Range Queries	range-aggregation			2.60	4.68
noaa	Range & Date Histogram	range-auto-date-histo			196.98	2,006.56
noaa	Range & Date Histogram	range-auto-date-histo-with-metrics			570.23	3,633.32
noaa	Range & Date Histogram	range-auto-date-histo-with-time-zone			198.41	2,031.60
noaa	Range & Date Histogram	range-date-histo			159.43	1,700.74
noaa	Range & Date Histogram	range-date-histo-with-metrics			427.54	2,814.44
noaa	Range Queries	range-numeric-significant-terms			1,122.67	2,496.51

Categorized Results

We zoom in on those operations that at least doubled the performance (i.e., took less than half as long to complete) over each engine when comparing median service times. In general, OpenSearch outperforms Elasticsearch with ascending sort and keyword term aggregation queries by more than 2x. In general, Elasticsearch outperforms OpenSearch with descending sort and date histogram queries by more than 2x.

Specific service time values in this section can be found in the Appendix. The following numbers can be referenced in the Executive Summary.

Sort

The goal of Sorting operations is to arrange values in either ascending or descending order based on specific fields.

- OpenSearch is **2.2x–4.7x faster** on Big5 ascending sort operations (`asc_sort_timestamp`, `asc_sort_timestamp_can_match_shortcut`, `asc_sort_timestamp_no_can_match_shortcut`, `asc_sort_with_after_timestamp`) and **200x–2,000x faster** on NYC Taxis sorting operations (`asc_sort_tip_amount`, `desc_sort_tip_amount`).
- Elasticsearch is **1.4x–2.8x faster** on Big5 descending sort operations (`desc_sort_timestamp`, `desc_sort_timestamp_can_match_shortcut`, `desc_sort_timestamp_no_can_match_shortcut`, `desc_sort_with_after_timestamp`).

Date histograms

Date histograms group documents based on date or timestamp fields into buckets according to a specified time interval. They are particularly useful for time-series analysis and visualizing data trends over time.

- OpenSearch is **38x–1,500x faster** on Big5 date histogram operations (`composite-date_histogram-daily`, `date_histogram_hourly_agg`, `date_histogram_minute_agg`).
- Elasticsearch is **3.7x faster** on Big5 date histogram operations (`range-auto-date-histo`, `range-auto-date-histo-with-metrics`) and **4.1x–4.9x faster** on NOAA operations (`date-histo-geohash-grid`, `date-histo-geotile-grid`, `date-histo-histo`, `date-histo-histo`).
- Elasticsearch is **5.7x–9.5x faster** on NOAA range and date histogram operations (`range-auto-date-histo`, `range-auto-date-histo-with-metrics`, `range-auto-date-histo-with-time-zone`, `range-date-histo`, `range-date-histo-with-metrics`).

Term Aggregations

Term Aggregations group documents based on specific field values. They create buckets where each bucket represents a unique term in the specified field, and documents are grouped into these buckets accordingly.

- OpenSearch is **120x-143x faster** on Big5 term aggregation operations (`keyword-terms`, `keyword-terms-low-cardinality`) and **3.6x–7.3x faster** on NOAA term aggregation operations (`keyword-terms`, `keyword-terms-low-cardinality`).
- Elasticsearch is **3.1x faster** on Big5 term aggregation operation (`cardinality-agg-high`) and **7.8x faster** on NOAA operation (`keyword-terms-low-cardinality-min`).

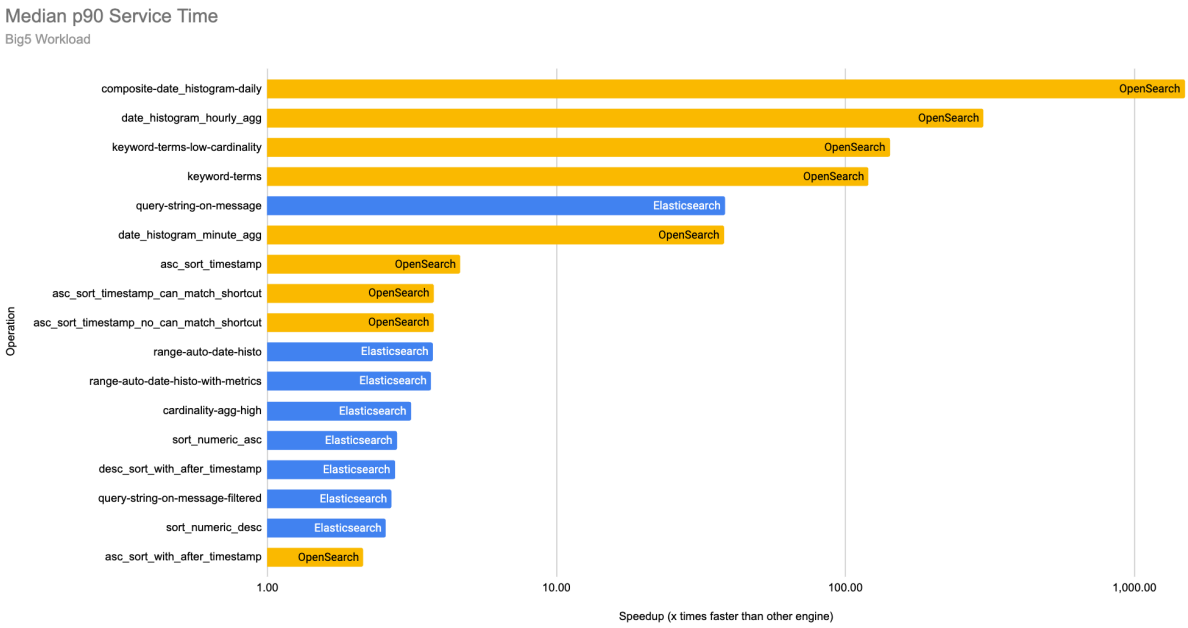
Others

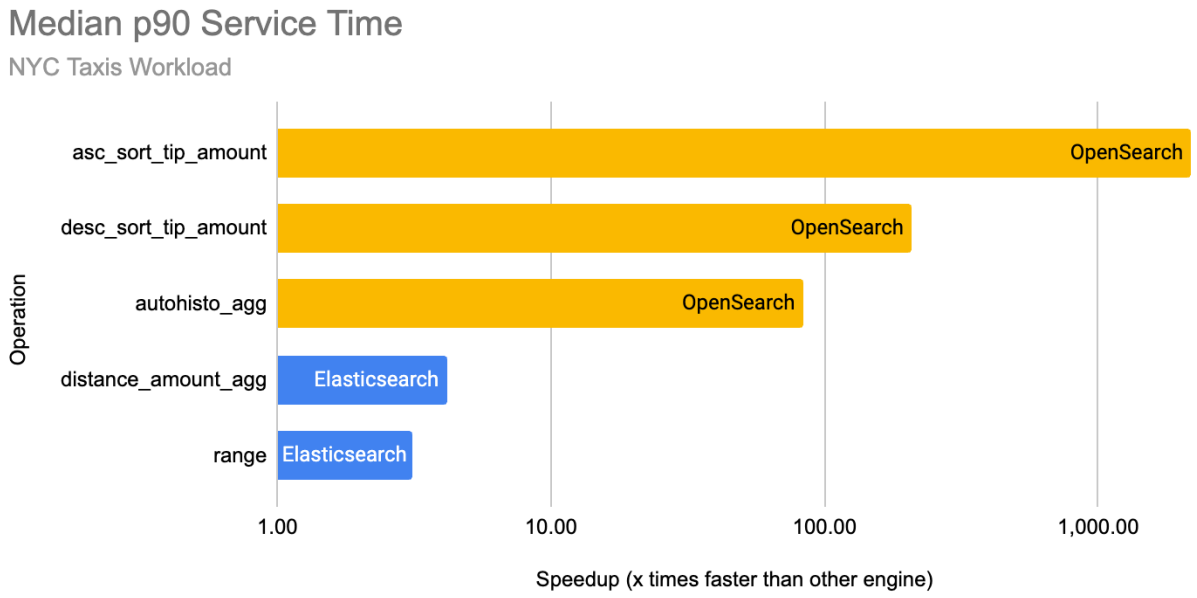
Other notable operation comparisons include:

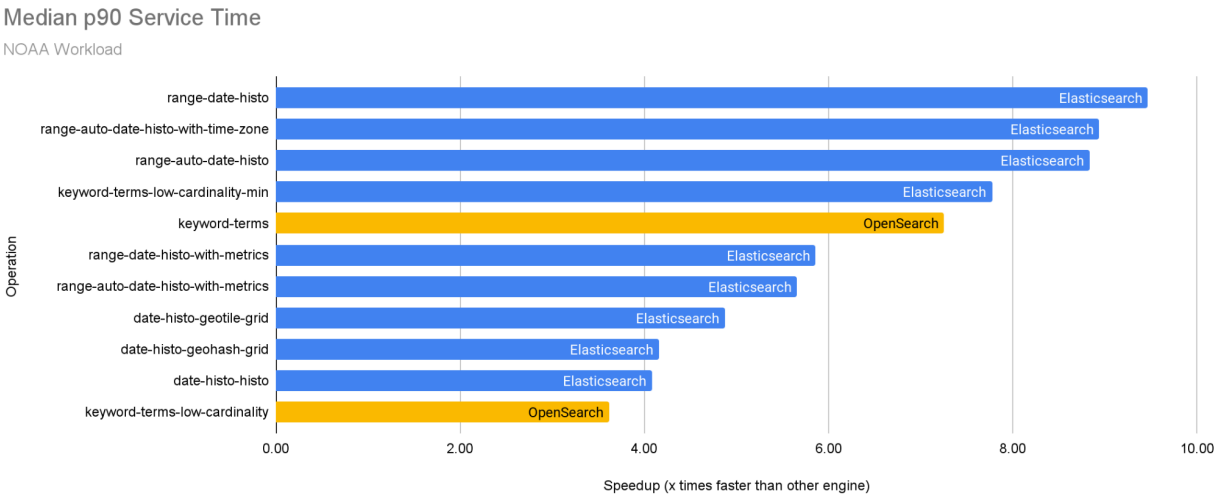
- OpenSearch is **84x faster** on a NYC Taxis aggregation operation (`autohisto_agg`).
- Elasticsearch is **2.7x–38x faster** on Big5 text querying operations (`query-string-on-message`, `query-string-on-message-filtered`).
- Elasticsearch is **3.1x faster** on NYC Taxis range operation (`range`).

- Elasticsearch is **4.2x faster** on NYC Taxis aggregation operation (`distance_amount_agg`).

The following graphs illustrate these results, where the y-axis represents an operation and the x-axis represents how many times faster an engine is over the other. From these graphs we can see that when OpenSearch is faster, it is significantly faster than Elasticsearch, while Elasticsearch outperforms OpenSearch on the NOAA workload, though by a smaller degree.




Median p90 Service Time
NYC Taxis Workload
asc_sort_tip_amount
desc_sort_tip_amount
autohisto_agg
distance_amount_agg
range
OpenSearch
OpenSearch
OpenSearch
Elasticsearch
Elasticsearch
Operation
1.00
10.00
100.00
1,000.00
Speedup (x times faster than other engine)


Median p90 Service Time
NOAA Workload
range-date-histo
range-auto-date-histo-with-time-zone
range-auto-date-histo
keyword-terms-low-cardinality-min
keyword-terms
range-date-histo-with-metrics
range-auto-date-histo-with-metrics
date-histo-geotile-grid
date-histo-geohash-grid
date-histo-histo
keyword-terms-low-cardinality
Elasticsearch
Elasticsearch
Elasticsearch
Elasticsearch
OpenSearch
Elasticsearch
Elasticsearch
Elasticsearch
Elasticsearch
Elasticsearch
OpenSearch
Operation
0.00
2.00
4.00
6.00
8.00
10.00
Speedup (x times faster than other engine)

Performance Trajectory

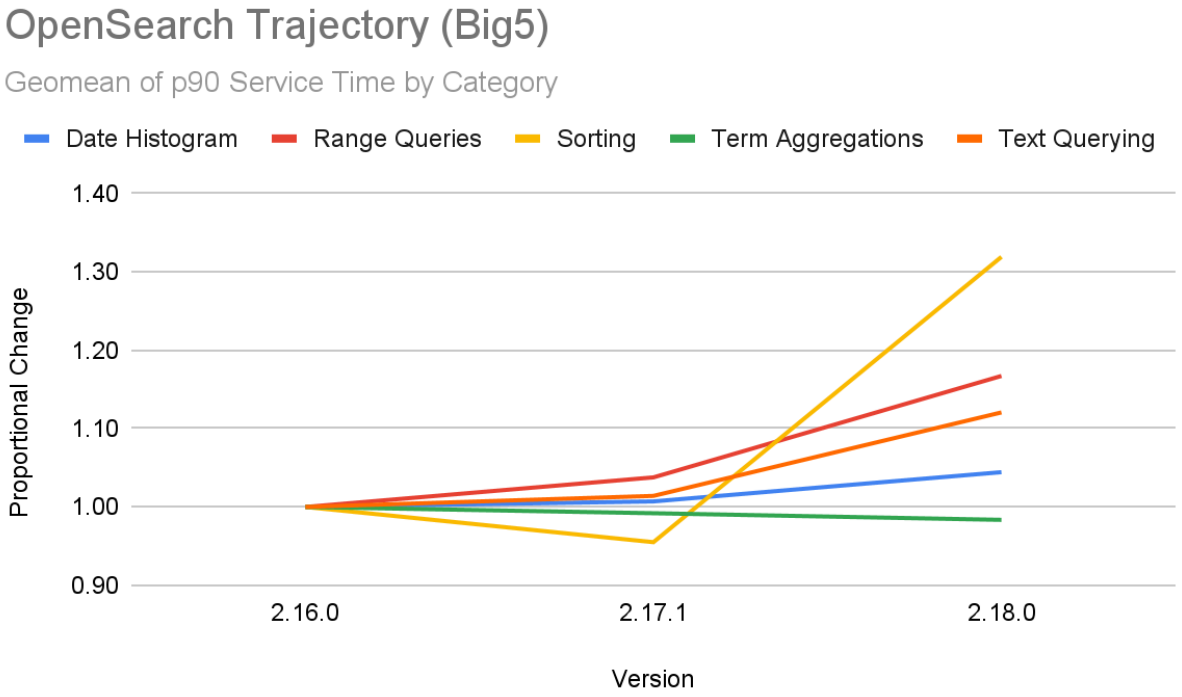
To measure performance trends over time, we ran OSB workloads on recent minor versions of OpenSearch (2.16.0, 2.17.1, 2.18.0) and Elasticsearch (8.13.4, 8.14.3, 8.15.0, 8.15.4). We examined how performance on workload operations improves and regresses over subsequent versions. We ran between two and eight tests for each version for four OSB workloads (Big5, NYC Taxis, PMC, and NOAA). The goal was to measure how performance improved or degraded between minor versions of each engine.

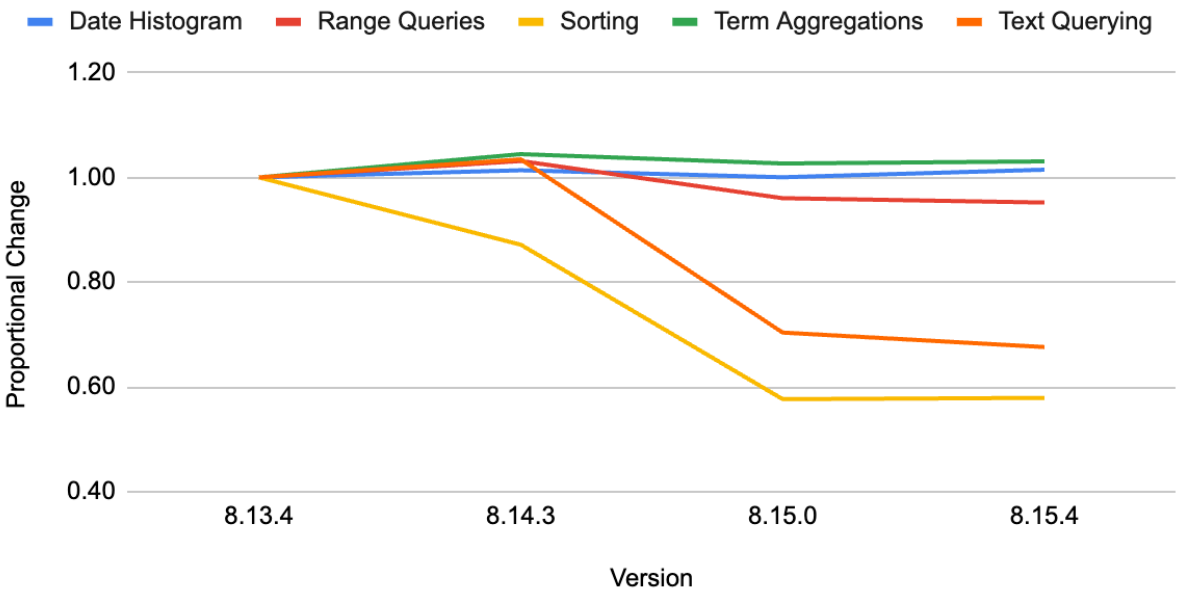
Note that, as our focus was mainly on the latest version of Elasticsearch, there were relatively fewer tests carried out on older releases. Older versions have a much lower number of tests and random variations could affect the conclusions drawn here (4 tests for 8.13.4, 4 for 8.14.3, 30 for 8.15.0, 75 for 8.15.4). See Validity of Results and Outlier Performance for more information.

Considering the geometric mean of the p90 service times grouped by workload, we observe the following:

- Big5:
  - OpenSearch 2.17.1 is faster than 2.16.0 by 0.6%
  - OpenSearch 2.18.0 is slower than 2.16.0 by 15%
  - Elasticsearch 8.15.4 is faster than 8.13.4 by 22%
- NYC Taxis:
  - OpenSearch 2.17.1 is slower than 2.16.0 by 35%
  - OpenSearch 2.18.0 is slower than 2.16.0 by 2%
  - Elasticsearch 8.15.4 is faster than 8.13.4 by 9.6%
- PMC:
  - OpenSearch 2.17.1 is slower than 2.16.0 by 4%
  - OpenSearch 2.18.0 is slower than 2.16.0 by 45%
  - Elasticsearch 8.15.4 is faster than 8.13.4 by 5.4%
- NOAA:
  - OpenSearch 2.17.1 is faster than 2.16.0 by 2%
  - OpenSearch 2.18.0 is slower than 2.16.0 by 0.1%
  - Elasticsearch 8.15.4 is slower than 8.13.4 by 5.5%
- NOAA Semantic Search:
  - OpenSearch 2.17.1 is faster than 2.16.0 by 0.4%
  - OpenSearch 2.18.0 is slower than 2.16.0 by 1.6%

For Big5 in particular, OpenSearch shows some regressions that were consistently observed in most test runs related to Sorting operations and some Range queries.


OpenSearch Trajectory (Big5)
Geomean of p90 Service Time by Category
Date Histogram
Range Queries
Sorting
Term Aggregations
Text Querying
1.40
1.30
1.20
1.10
1.00
0.90
Proportional Change
2.16.0
2.17.1
2.18.0
Version

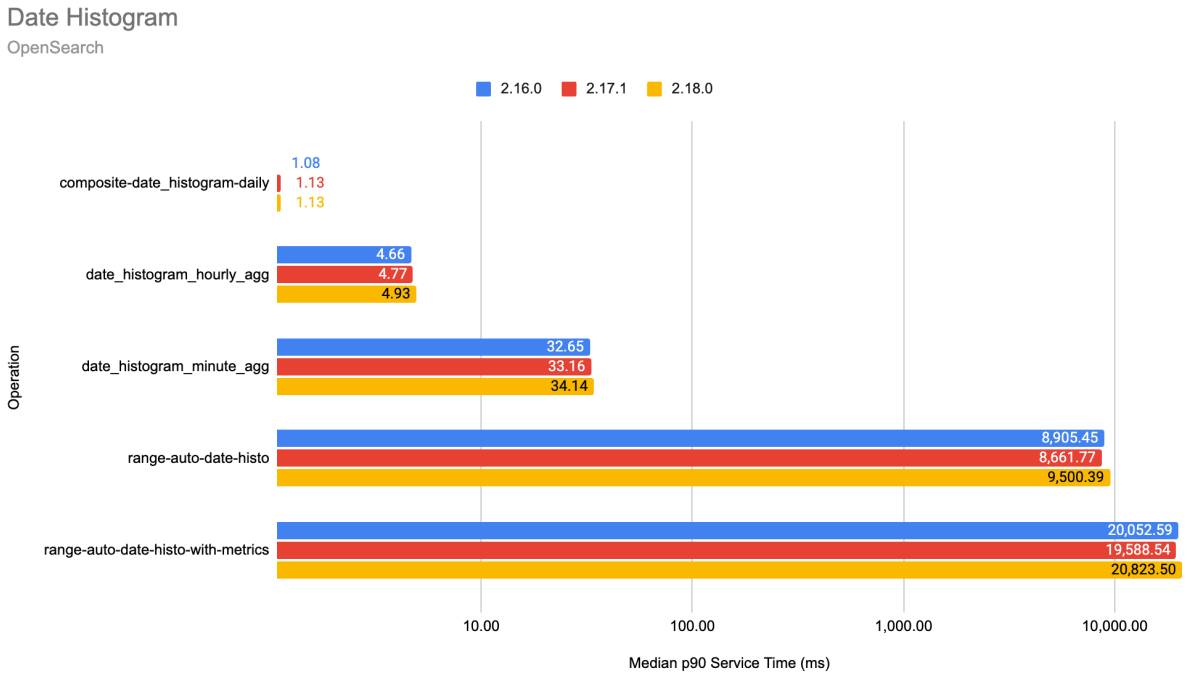

Elasticsearch Trajectory (Big5)
Geomean of p90 Service Time by Category
Date Histogram
Range Queries
Sorting
Term Aggregations
Text Querying
1.20
1.00
0.80
0.60
0.40
Proportional Change
8.13.4
8.14.3
8.15.0
8.15.4
Version

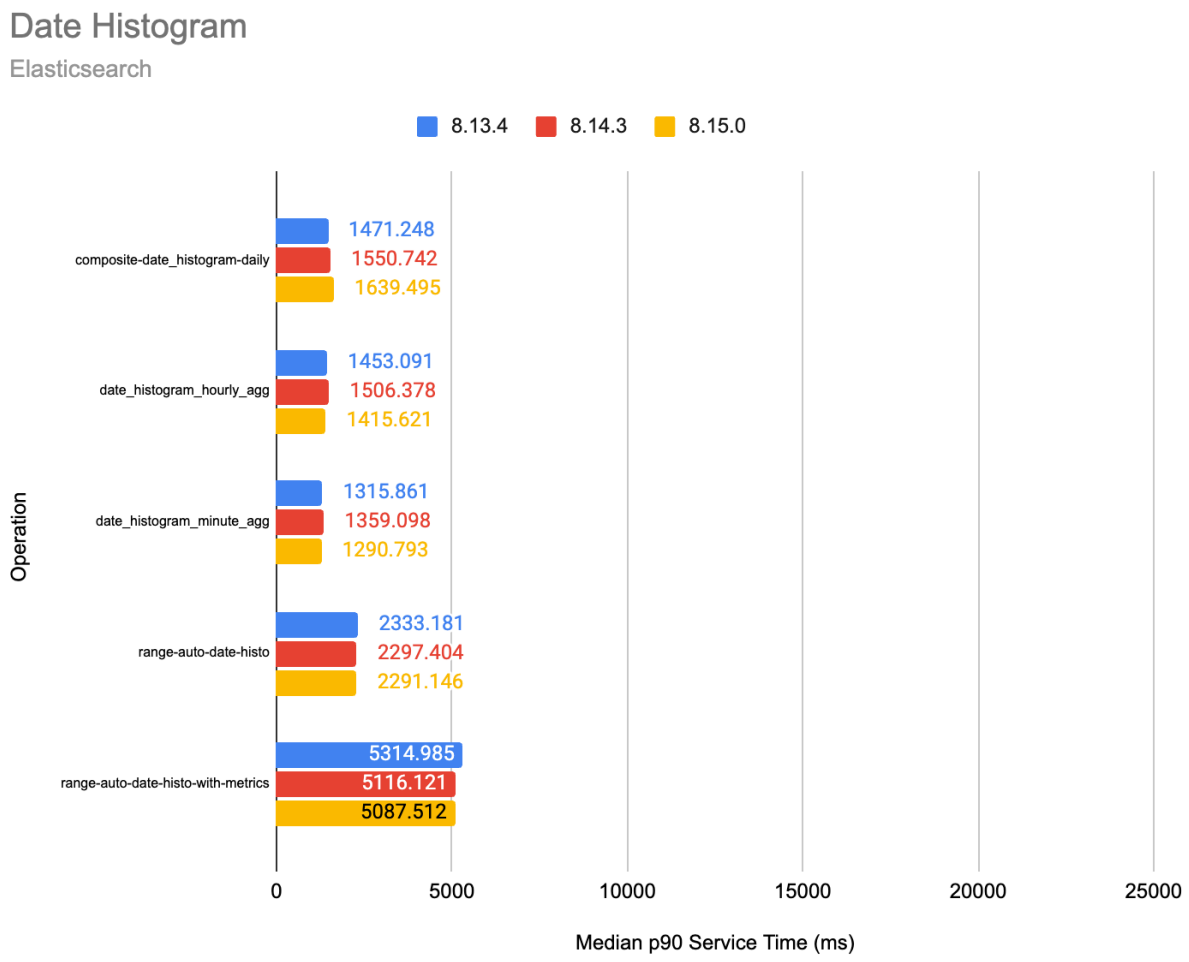
Categorized Results

Big5 Date Histogram

OpenSearch remains consistent across versions when considering Date Histogram operations, except for 2.18.0, which shows slightly slower results for a few operations.

Elasticsearch has small but consistent improvements across versions on most date histogram operations.

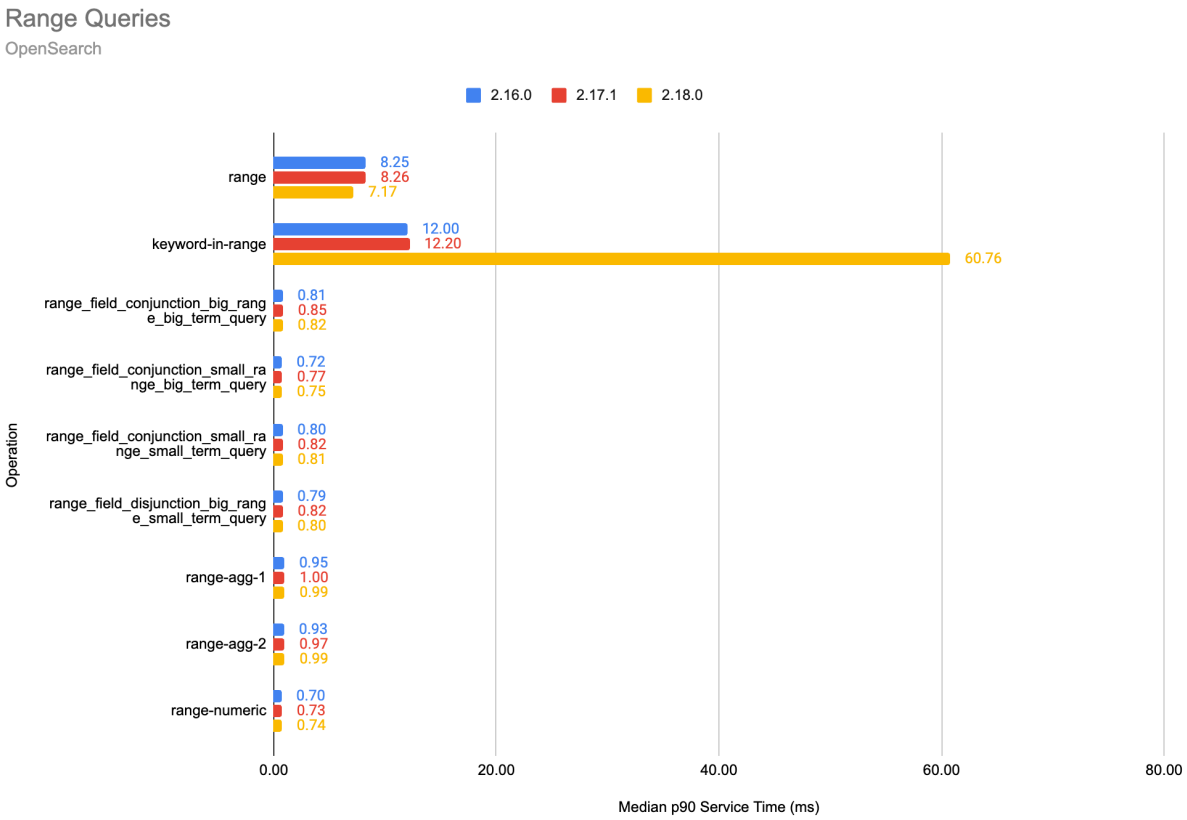


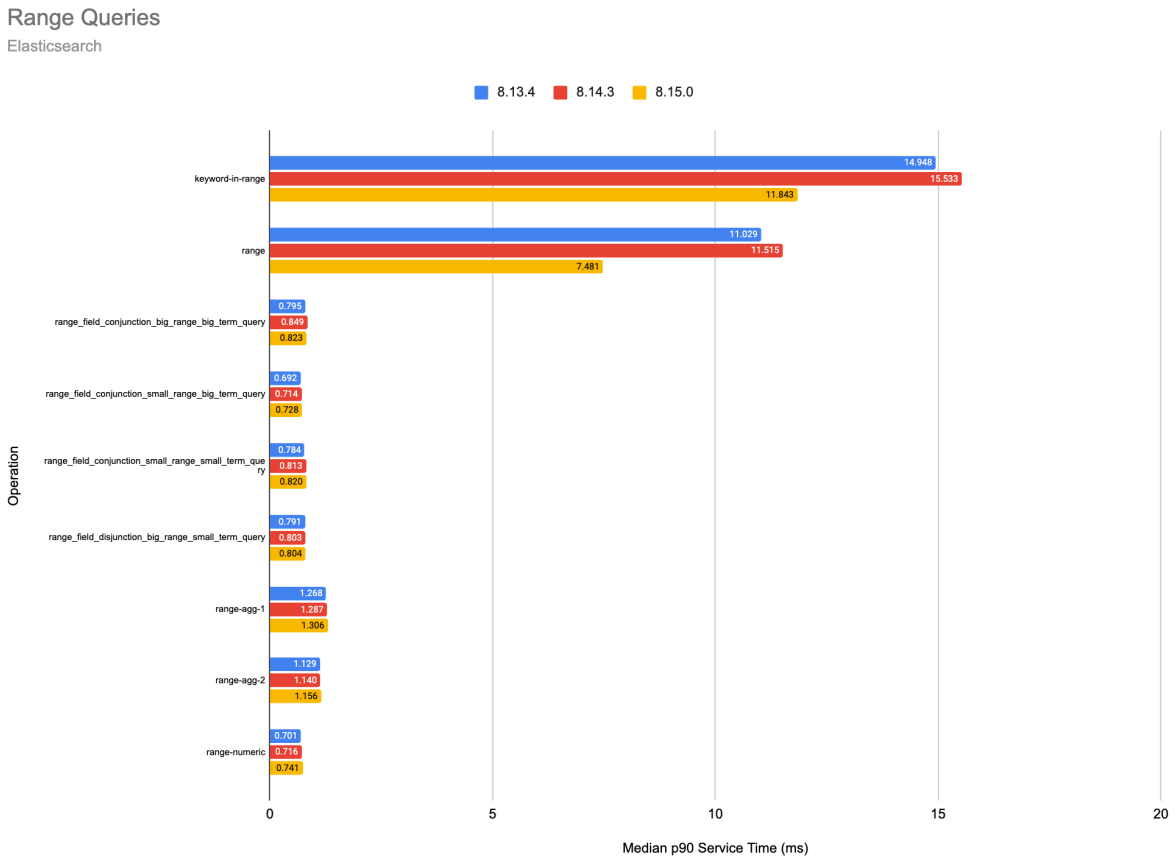


Big5 Range Queries

OpenSearch has a 5x slowdown in version 2.18.0 in the `keyword-in-range` operation across all tests we ran.

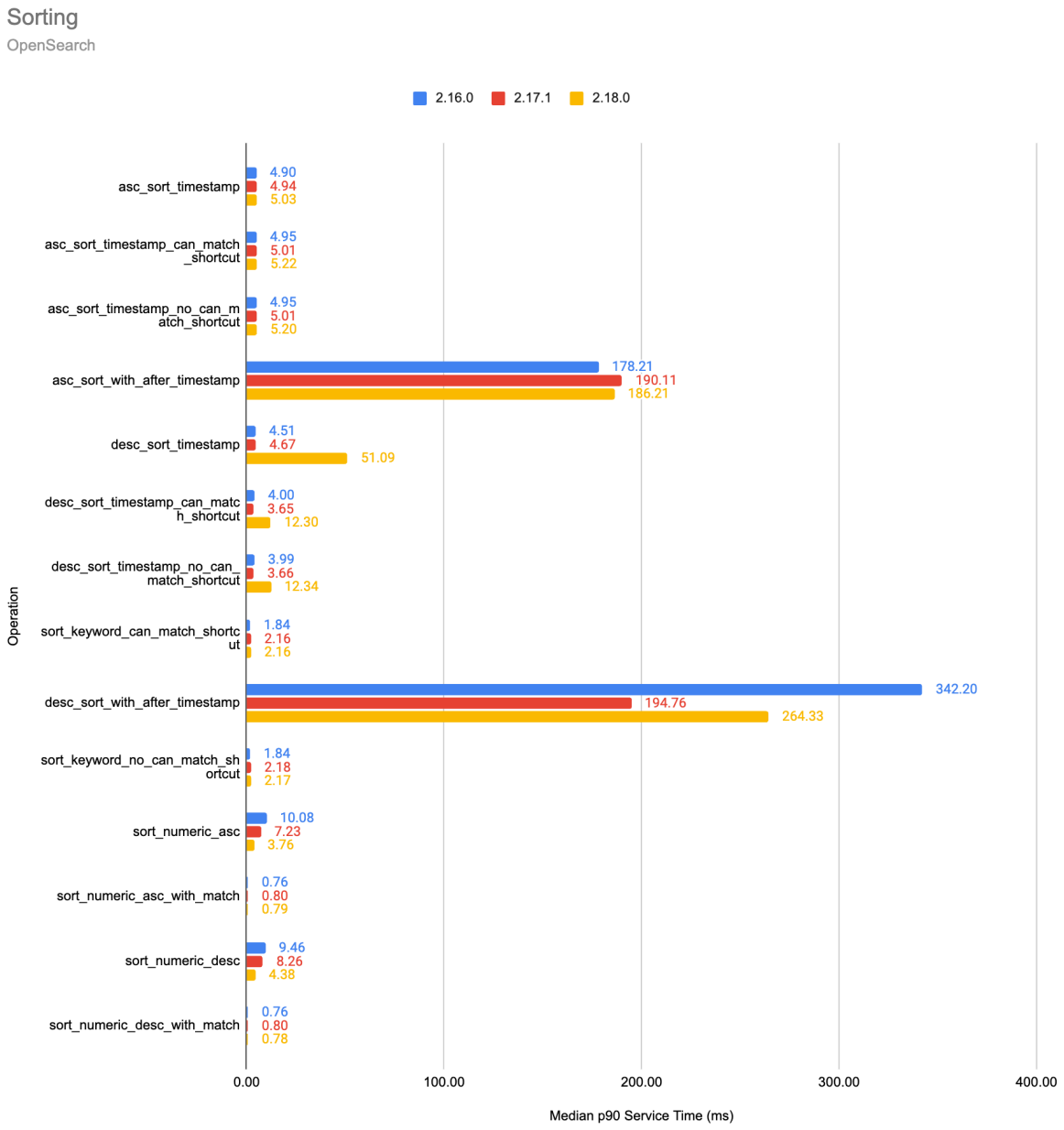
Elasticsearch has improvements (~20%-30% faster) in 8.15.0 compared to previous versions in `range` and `keyword-in-range` while remaining stable on the other operations.


Range Queries
OpenSearch
2.16.0
2.17.1
2.18.0
range
keyword-in-range
range_field_conjunction_big_range_big_term_query
range_field_conjunction_small_range_big_term_query
range_field_conjunction_small_range_small_term_query
range_field_disjunction_big_range_small_term_query
range-agg-1
range-agg-2
range-numeric
8.25
8.26
7.17
12.00
12.20
60.76
0.81
0.85
0.82
0.72
0.77
0.75
0.80
0.82
0.81
0.79
0.82
0.80
0.95
1.00
0.99
0.93
0.97
0.99
0.70
0.73
0.74
Operation
0.00
20.00
40.00
60.00
80.00
Median p90 Service Time (ms)

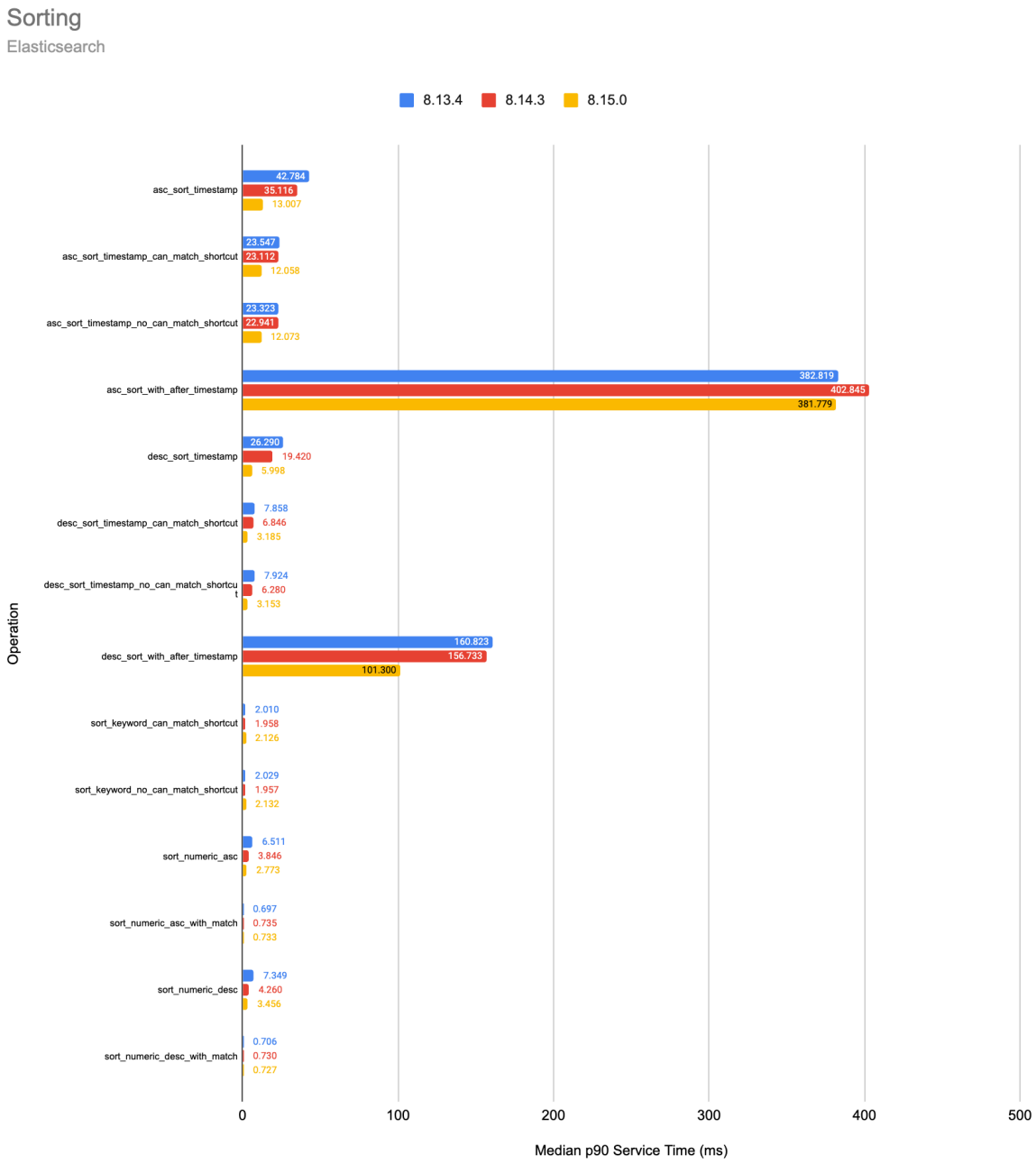


Big5 Sorting

OpenSearch has a mix of results in the Sorting operations across versions, with some Big5 operations (`sort_numeric_asc` and `sort_numeric_desc`) becoming 2x faster in 2.18.0 compared to the previous version 2.17.1, while other Big5 operations (`desc_sort_timestamp`, `desc_sort_timestamp_no_can_match_shortcut`, `desc_sort_timestamp_can_match_shortcut`) have a considerable slowdown up to 11x.

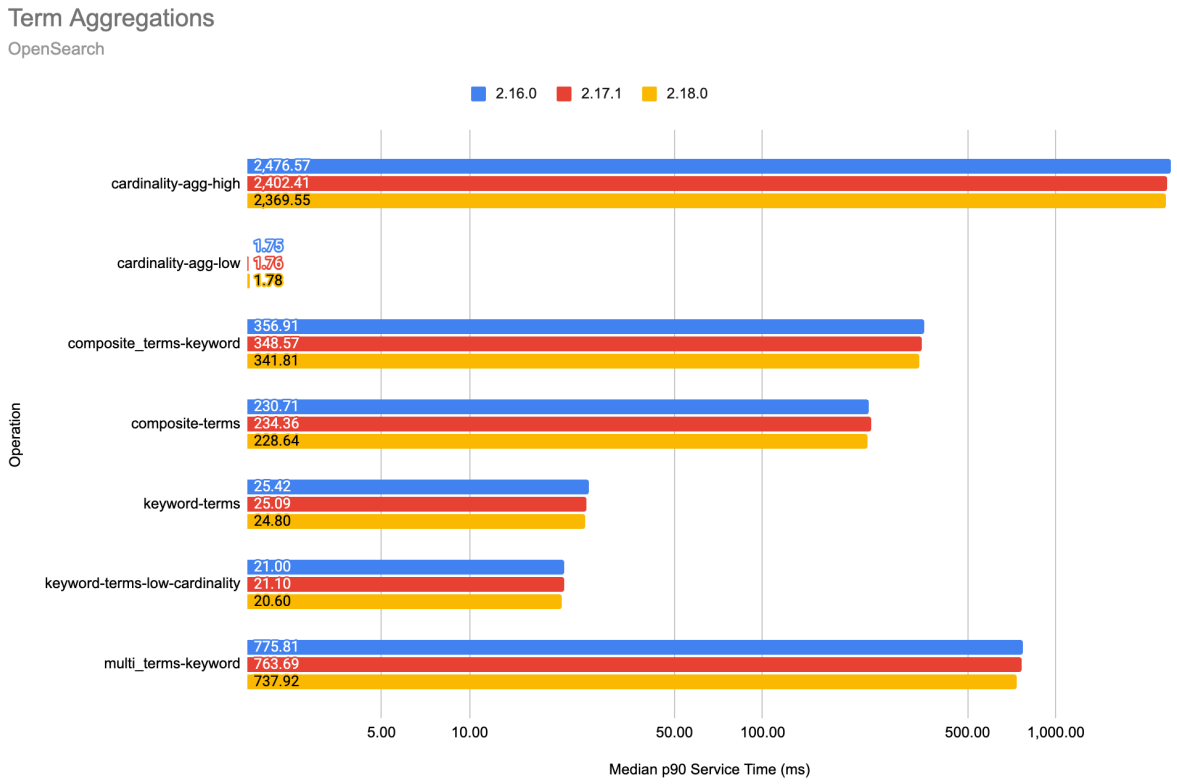


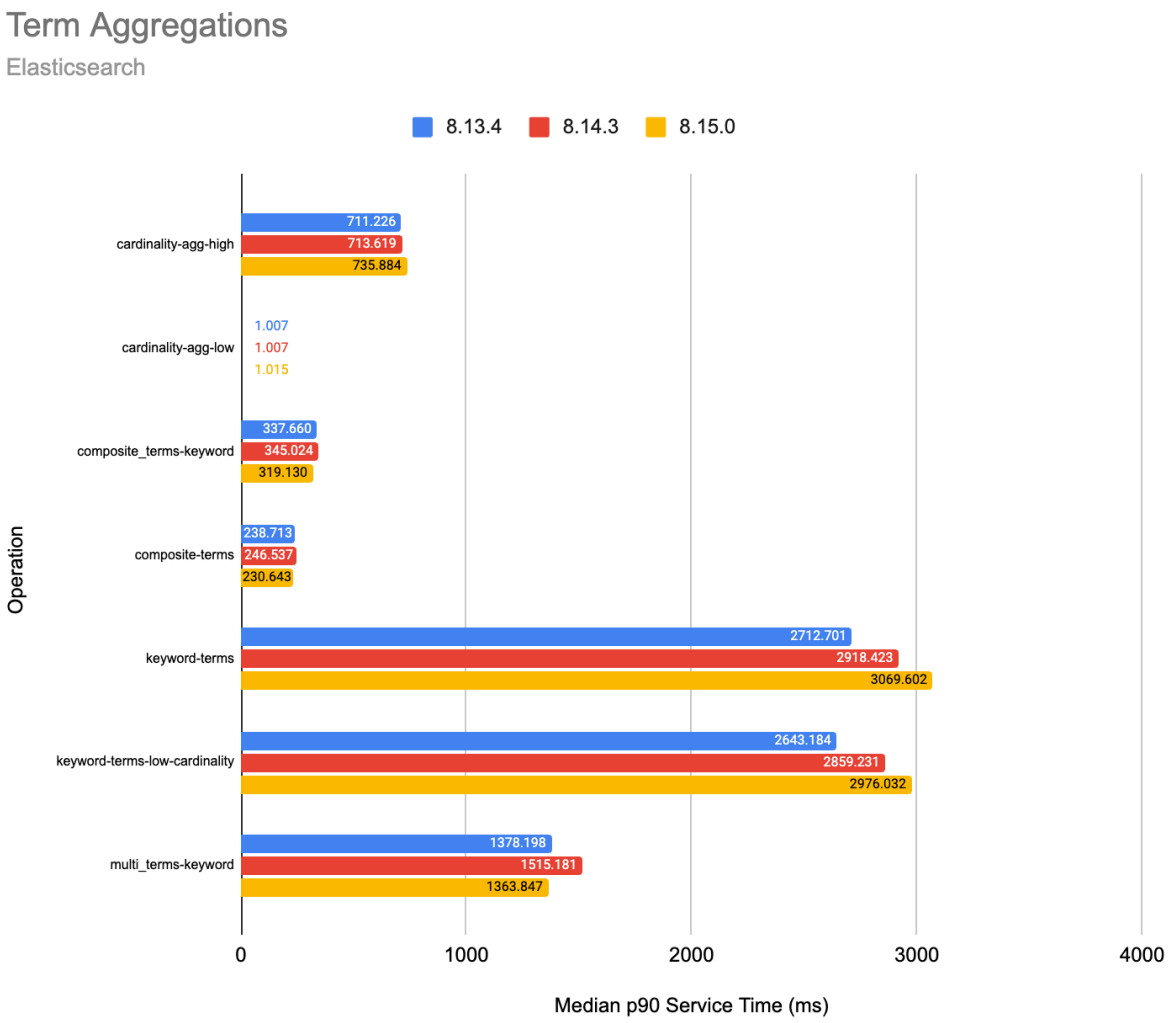
Elasticsearch 8.15.0 improved over 8.14.3 on several Big5 operations (`desc_sort_timestamp`, `desc_sort_timestamp_no_can_match_shortcut`, `desc_sort_timestamp_can_match_shortcut`, `desc_sort_with_after_timestamp`, `asc_sort_timestamp`) by 1.5x–2.7x.



Big5 Term Aggregations

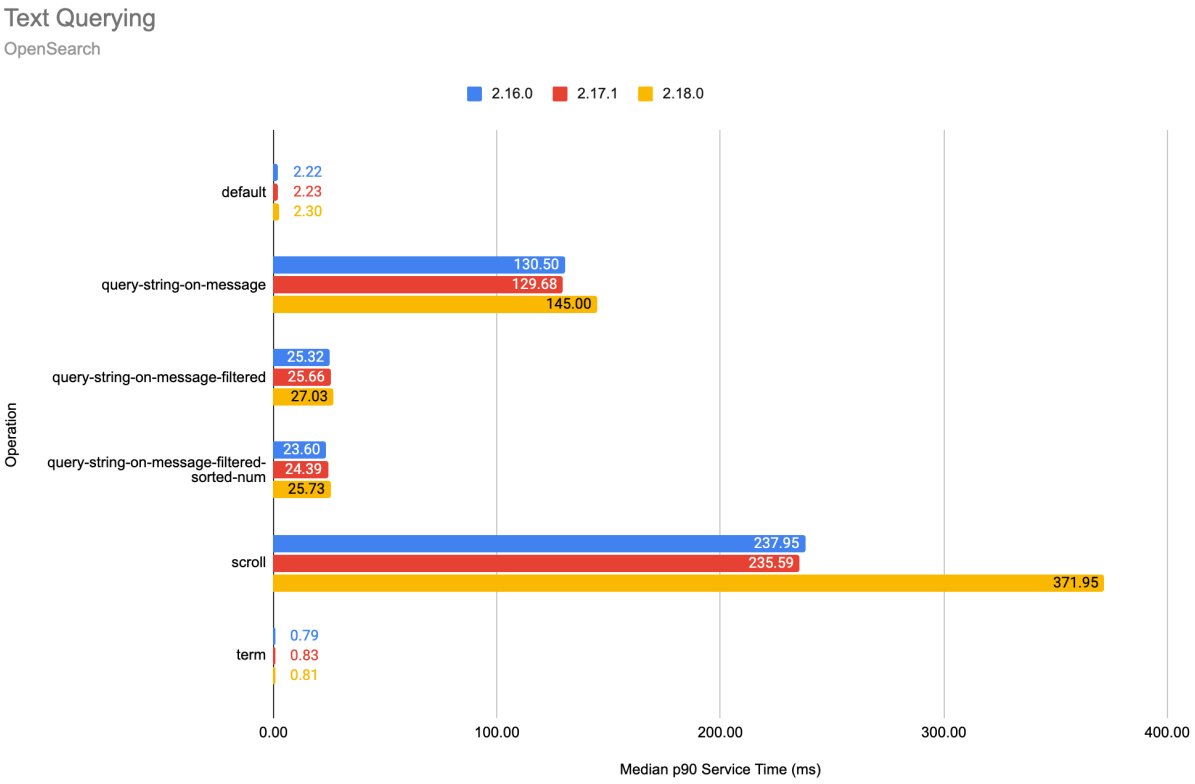
Overall, both engines remained relatively stable on this set of operations, with OpenSearch improving slightly on the `cardinality-agg-high` operations. Elasticsearch had noticeable degradations in performance for `keyword-terms` and `keyword-terms-low-cardinality` operations.

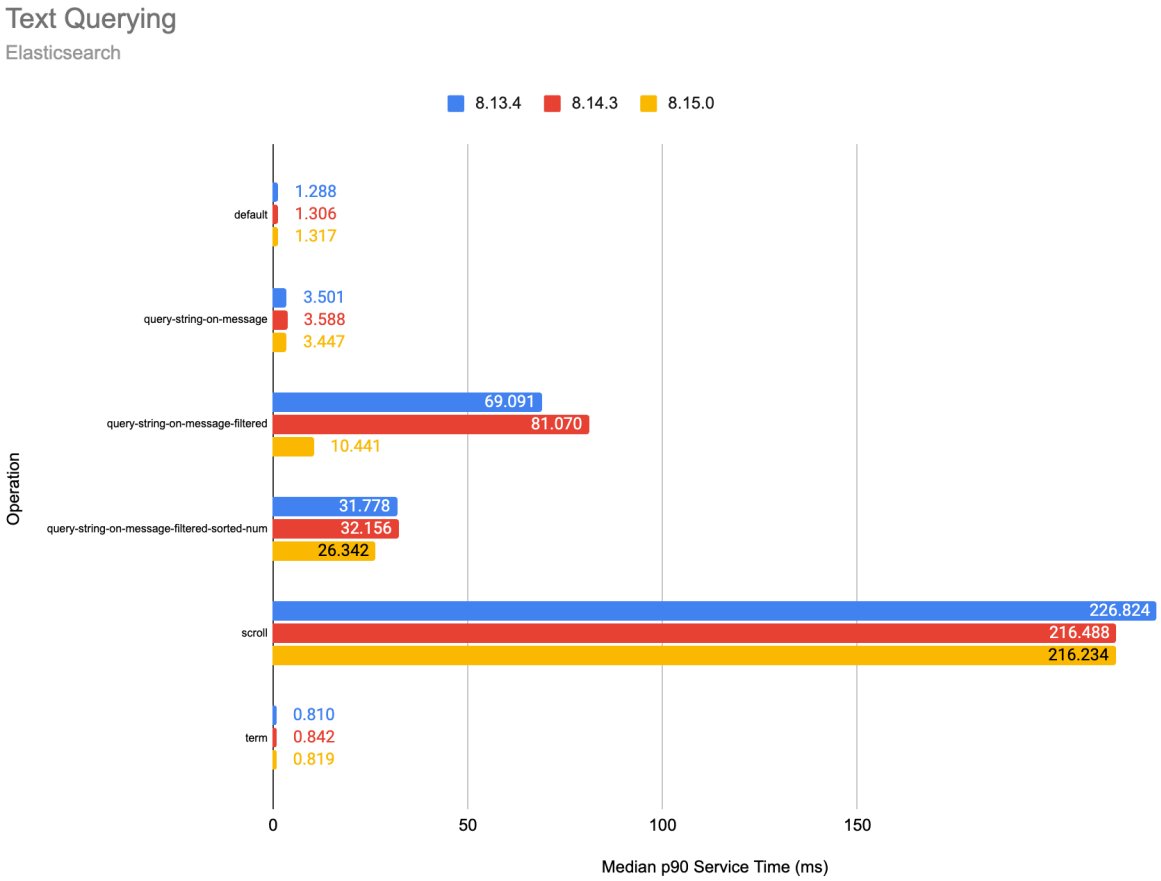

Term Aggregations
OpenSearch
2.16.0
2.17.1
2.18.0
cardinality-agg-high
2,476.57
2,402.41
2,369.55
cardinality-agg-low
1.75
1.76
1.78
composite_terms-keyword
356.91
348.57
341.81
composite-terms
230.71
234.36
228.64
keyword-terms
25.42
25.09
24.80
keyword-terms-low-cardinality
21.00
21.10
20.60
multi_terms-keyword
775.81
763.69
737.92
Operation
5.00
10.00
50.00
100.00
500.00
1,000.00
Median p90 Service Time (ms)



Big5 Text Querying

There were noticeable degradations in performance for OpenSearch Text Queries (particularly for `scroll` and `query-string-on-message` operations). However, Elasticsearch shows a **6.6x-7.8x** speedup on the `query-string-on-message-filtered` operation in version 8.15.0 compared to previous versions.


Text Querying
OpenSearch
2.16.0
2.17.1
2.18.0
default
2.22
2.23
2.30
query-string-on-message
130.50
129.68
145.00
query-string-on-message-filtered
25.32
25.66
27.03
query-string-on-message-filtered-sorted-num
23.60
24.39
25.73
scroll
237.95
235.59
371.95
term
0.79
0.83
0.81
Operation
0.00
100.00
200.00
300.00
400.00
Median p90 Service Time (ms)



Vectorsearch

This section reports results from our vectorsearch workload (separately from the other four workloads) because of the distinct nature of this workload and the changes we made to its configuration.

OpenSearch supports three vectorsearch engines: NMSLIB, FAISS, and Lucene. Elasticsearch supports only Lucene. Any reported values in the charts below for Elasticsearch are when it is run with Lucene. For brevity, we specify a search engine and the vectorsearch engine used in the format: search engine (vector engine).

Only one task was tested in the vectorsearch workload: `prod-queries`.

Standard Configuration

The standard configuration refers to a configuration described in the Evaluation Setup that we used for our nightly benchmarks. We tested Vectorsearch with the standard configuration from December 5, 2024, to December 16, 2024.

Comparing each OpenSearch 2.17.1 vector engine against Elasticsearch (Lucene) 8.15.4 across OpenSearch versions yielded the following findings:

- OpenSearch (FAISS) was **13.8% faster**
- OpenSearch (NMSLIB) was **11.3% faster**
- OpenSearch (Lucene) was **258.2% slower**

The default vector engine for OpenSearch is FAISS for >=2.18.0 and NMSLIB for <2.18.0. We compared each OpenSearch vector engine against Elasticsearch (Lucene) (instead of only the default vector engine) so that we could understand how each available OpenSearch configuration would perform. We also compared the OpenSearch vector engines against each other to make the performance differences between them more explicit. Averaging the results of each OpenSearch version (2.16.0, 2.17.1, 2.18.0) yielded the following results:

- OpenSearch (FAISS) averaged **7.48 ms**
- OpenSearch (NMSLIB) averaged **7.70 ms**
- OpenSearch (Lucene) averaged **28.55 ms**

The slower performance of OpenSearch (Lucene) was driven by slow outlier runs. Although the average was 28.55 ms, the median value was 16.54 ms. The slowest outlier was 252.34 ms. As we discuss in the following Additional Configuration Experiments section, we are able to reduce outliers in OpenSearch (Lucene) in an experiment with a modified JVM Heap size and snapshot process.

Sparklines

Similar to the above Section, we created two sets of sparkline plots to visualize all of the data for p90 service times. Lower values are better (i.e., faster).

First, we show the *variation* in service times (i.e., not using the same scale) for each engine.

Workload	Category	Operation	Workload Subtype	OpenSearch 2.16.0	OpenSearch 2.17.1	OpenSearch 2.18.0	Elasticsearch 8.15.4
vectorsearch	ML	prod-queries	faiss-cohere-10m				
vectorsearch	ML	prod-queries	lucene-cohere-10m				
vectorsearch	ML	prod-queries	nmslib-cohere-10m				

Next, we show the *same scale* for service times, comparing recent versions of OpenSearch (2.17.1) and Elasticsearch (8.15.4). The maximum and minimum values of the sparklines are displayed to the right of each plot.

Workload	Category	Operation	Workload Subtype	OpenSearch 2.17.1	Elasticsearch 8.15.4	Min	Max
vectorsearch	ML	prod-queries	faiss-cohere-10m			7.22	9.17
vectorsearch	ML	prod-queries	lucene-cohere-10m			8.39	129.49
vectorsearch	ML	prod-queries	nmslib-cohere-10m			7.53	9.17

Additional Configuration Experiments

We performed additional experiments to explore other vectorsearch configuration scenarios.

In response to the outliers we observed in our nightly runs, we ran a modified configuration with a lower JVM Heap size (32GB) and modified snapshot process. The modified snapshot process flushed the index, refreshed the index, and waited for merges to complete before taking a snapshot. With this modified configuration, we did not observe outlier runs in OpenSearch (Lucene). OpenSearch (Lucene) had an average p90 Service Time of **16.4 ms** across versions, which matches the median value observed in our standard runs. Additionally, we observed that OpenSearch FAISS and NMSLIB were faster than Elasticsearch for all versions except for two configurations: OpenSearch (FAISS) 2.16.0 was 1.6% slower than Elasticsearch, and OpenSearch (NMSLIB) 2.17.1 was 10.1% slower. The slower NMSLIB results were driven by outlier runs that are 2x slower than the median. We observed a speedup in Elasticsearch. We also tested a development version of OpenSearch 2.19.0 (2.19.0-nightly-10607). We observed a **32.9% speedup** from OpenSearch (Lucene) 2.18.0 to OpenSearch (Lucene) 2.19.0.

We adapted this modified configuration to another experiment that investigated concurrent search on an index that was not force-merged. Since the index was not force-merged, it had many segments instead of the default of one segment for the vectorsearch workload. We enabled concurrent search in the configuration for OpenSearch (while Elasticsearch enables this feature by default). We also set the index setting `index.knn.advanced.approximate_threshold` [2] to 0 in OpenSearch FAISS and NMSLIB versions >=2.18.0 to enable regeneration of vectorsearch data structures. Prior to 2.18.0, this option did not exist but was effectively enabled. We tested OpenSearch 2.19-development in this experiment as well. We observed the following:

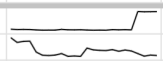
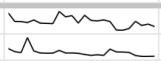
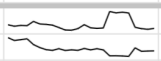
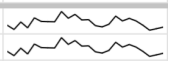
- Excluding OpenSearch 2.19-development, Elasticsearch (Lucene) was **38.4%–77.4% faster** than all OpenSearch vector engines.
- For OpenSearch 2.19-development, we observed a **46.6% speedup** for OpenSearch (Lucene) and a **22.3% speedup** for OpenSearch (FAISS) over the fastest respective results for a currently released OpenSearch version. We did not observe a speedup for OpenSearch (NMSLIB). The OpenSearch (FAISS) speedup was driven by the 2.19-development results not having outliers. While OpenSearch 2.19-development remained slower than Elasticsearch across vector engines, the speedups narrowed the gap.
    - It is important to note that we observed outliers in this experiment, which influenced the results. We observed fewer outliers for OpenSearch 2.19-development, which in particular drove the FAISS speedup. However, given our sample size of 16 runs per 2.19-development configuration, it is

possible that the result of fewer outliers for OpenSearch 2.19-development was due to chance.

Sparklines: Force Merge Enabled

Similar to the above Section, we created two sets of sparkline plots to visualize all of the data for p90 service times. Lower values are better (i.e., faster).

First, we show the *variation* in service times (i.e., not using the same scale) for each engine.

Workload	Category	Operation	Workload Subtype	OpenSearch 2.16.0	OpenSearch 2.17.1	OpenSearch 2.18.0	OpenSearch 2.19.0	Elasticsearch 8.15.4
vectorsearch	ML	prod-queries	faiss-cohere-10m					
vectorsearch	ML	prod-queries	lucene-cohere-10m					
vectorsearch	ML	prod-queries	nmslib-cohere-10m					

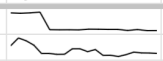
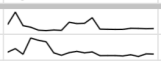
Next, we show the *same scale* for service times, comparing the latest versions of OpenSearch (2.17.1 and 2.19-development) and Elasticsearch (8.15.4). The maximum and minimum values of the sparklines are displayed to the right of each plot.

Workload	Category	Operation	Workload Subtype	OpenSearch 2.17.1	OpenSearch 2.19.0	Elasticsearch 8.15.4	Min	Max
vectorsearch	ML	prod-queries	faiss-cohere-10m				7.10	8.85
vectorsearch	ML	prod-queries	lucene-cohere-10m				8.43	19.69
vectorsearch	ML	prod-queries	nmslib-cohere-10m				7.19	15.91

Sparklines: Force Merge Disabled

These sparklines were created similarly, except for measuring performance when force merge was disabled.

First, we show the *variation* in service times (i.e., not using the same scale) for each engine.

Workload	Category	Operation	Workload Subtype	OpenSearch 2.16.0	OpenSearch 2.17.1	OpenSearch 2.18.0	OpenSearch 2.19.0	Elasticsearch 8.15.4
vectorsearch	ML	prod-queries	faiss-cohere-10m					
vectorsearch	ML	prod-queries	lucene-cohere-10m					
vectorsearch	ML	prod-queries	nmslib-cohere-10m					

Next, we show the *same scale* for service times, comparing the latest versions of OpenSearch and Elasticsearch.

Workload	Category	Operation	Workload Subtype	OpenSearch 2.17.1	OpenSearch 2.19.0	Elasticsearch 8.15.4	Min	Max
vectorsearch	ML	prod-queries	faiss-cohere-10m				12.45	73.09
vectorsearch	ML	prod-queries	lucene-cohere-10m				12.45	129.39
vectorsearch	ML	prod-queries	nmslib-cohere-10m				12.45	35.51

Performance Validation

While we report results from a two-week period above (chosen at the end of our engagement), we ran roughly nine weeks of nightly tests to ensure that we saw consistent performance. Interestingly, we observed that both OpenSearch and Elasticsearch performance dipped significantly for various tests, despite the configuration not changing. Within the context of our experimental setup, we ensured we controlled the major system resources and fixed configuration parameters that could affect performance.

Validation of Results

To validate our results, we calculated the p-value (the likelihood that a measured difference is due to chance) and statistical power (the likelihood of being able to measure a difference, if there is any, and consequently its accuracy) for our dataset. We used a p-value of 0.05 as our threshold (as is recommended [1]).

We observed that some operations (mainly those whose service time values were less than one millisecond) had a statistical power lower than our chosen threshold of 90%, despite having a low p-value: this signifies that a statistically significant difference could be detected, but its magnitude could not be observed reliably enough. Therefore, we chose to execute additional runs of those operations in an attempt to increase their statistical power. However, we note that users are likely more concerned about the performance of longer-running queries compared to those that complete within a few milliseconds.

These performance variation experiments ran a subset of tasks in each workload many times in order to statistically determine performance differences between OpenSearch versions and between OpenSearch and Elasticsearch. The subset was chosen to contain tasks where the measured p-value was lower than 5% (indicating a high chance of real performance differences between OpenSearch and Elasticsearch), and the statistical power was lower than 90% (indicating that an accurate measurement of the difference in performance would need more samples). The reasoning behind this choice is that with low statistical power, it is not possible to have a precise picture of the difference in performance and that collecting more samples would improve the situation.

These are the chosen tasks from the Big5 workload:

- `sort_numeric_desc_with_match`
- `range_field_conjunction_small_range_big_term_query`
- `sort_numeric_asc_with_match`
- `sort_keyword_no_can_match_shortcut`
- `sort_keyword_can_match_shortcut`
- `range-agg-2`

The preexisting infrastructure used for the nightly runs had to be slightly modified to allow the spawning of multiple instances of the benchmarks in parallel.

The original Terraform scripts were meant to be completely self-contained, creating all of the necessary AWS resources from scratch and automatically tearing them down once the benchmarking was over. This made the benchmarking workflow simple but presented challenges when scaled out since some AWS resources have lower quotas than others (VPC was the biggest offender). Instead of creating these from scratch, we made the infrastructure rely on preexisting resources that can be created ahead of time and specified at the moment of spawning the infrastructure.

Additionally, running this large number of tests in parallel exposed some reliability issues in the scripts that have now been fixed and also benefit the regular nightly runs.

The additional samples we collected confirmed that any statistically significant difference between the performance characteristics of OpenSearch and Elasticsearch in the cases of the tasks noted above is inconsequential. The growing sample size given by the nightly runs still helps provide more accurate figures when comparing the various OpenSearch versions under test: the results are clear enough to be used as a guideline for investigating improvements and regressions in performance across OpenSearch versions 2.16, 2.17, and 2.18.

In conclusion, these results validated the performance results discussed in this report.

Outlier Performance

We observe slow outlier runs for p90 service time for both OpenSearch and Elasticsearch. These outlier runs are much slower than the average values for p90 service time and occur intermittently. We investigated outlier runs for all workloads, excluding noaa-semantic-search and vectorsearch.

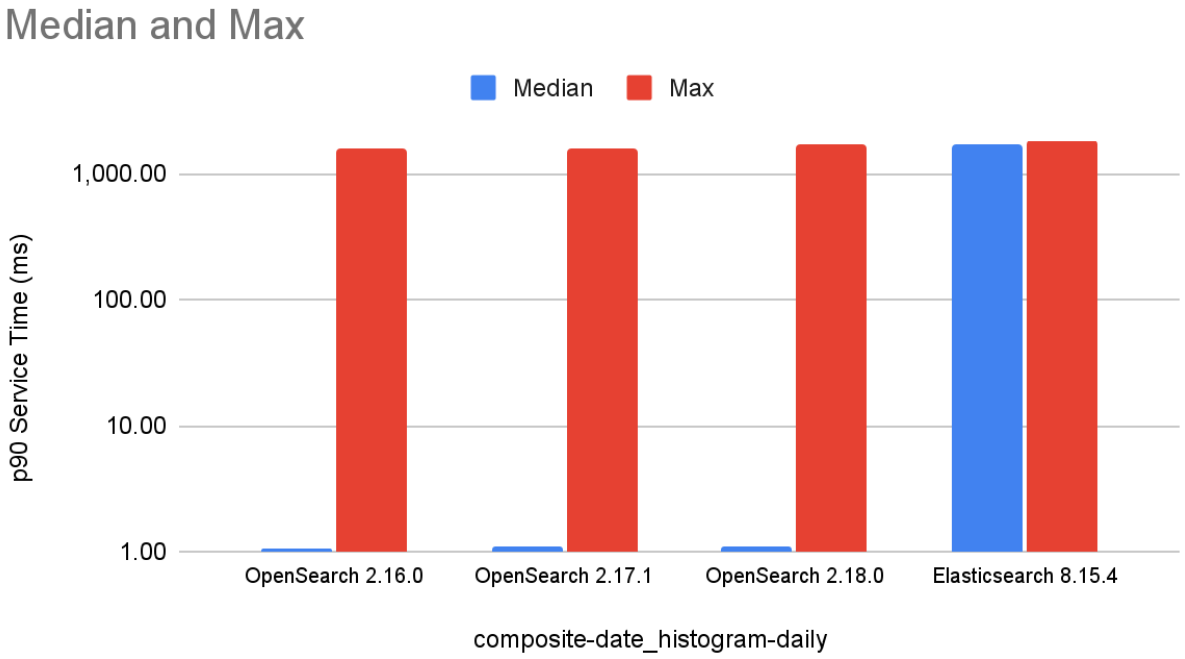
We can quantify how extreme outliers are by the ratio of the maximum service time over the median service time. **With this ratio, OpenSearch has outliers that are more extreme than Elasticsearch.** The tasks that with the most extreme ratios for each OpenSearch and Elasticsearch version were:

- OpenSearch 2.16.0: **1473x** for `composite-date_histogram-daily`
- OpenSearch 2.17.1: **1412x** for `composite-date_histogram-daily`
- OpenSearch 2.18.0: **1520x** for `composite-date_histogram-daily`
- Elasticsearch 8.15.4: **43x** for `query-string-on-message`

We count how many tasks have outlier runs, using the criterion of a run with a value that is more than twice as slow as the median. **We found that Elasticsearch has more outliers than OpenSearch**:

- OpenSearch 2.16.0: 11 outlier tasks out of 98
- OpenSearch 2.17.1: 11 outlier tasks out of 98
- OpenSearch 2.18.0: 12 outlier tasks out of 98
- Elasticsearch 8.15.4: 19 outlier tasks out of 98

The following graphs display the tasks that are the most extreme outliers for Elasticsearch (`query-string-on-message`) and OpenSearch (`composite-date_histogram-daily`).



Instance Variation

We observed variations in nightly runs with the same configuration and investigated how to control it. We found that the variance can be substantially reduced by reusing the same instances. The variance experiments informed our final experimental design.

We created an experiment that repeatedly ran the Big5 workload in several different scenarios to compare how much results vary in each scenario. We thought the variance could be caused by variance in the underlying hardware, even though the instances were the same type. To test this hypothesis, we tested repeatedly running Big5 on: different instances; the same instances (with a restored snapshot each time); and different instances that were deployed to the same dedicated host.

We found that running Big5 on the same instance had the least variance. The scenarios of different instances and different instances with the same dedicated host had higher variance. We calculated the geometric mean of Relative Standard Deviation (RSD) for p90 service time for tasks in each scenario.

Scenario	OpenSearch RSD Geometric Mean (%)	Elasticsearch RSD Geometric Mean (%)
Same instance	1.5%	1.5%
Same dedicated host	3.1%	2.5%
Different instance	3.7%	5.0%

It is notable that running instances on the same dedicated host did not control the variance like running on the same instance did. This indicates that variation in the underlying hardware cannot explain the variance. Additionally, we recorded hardware information from all instances to determine if we could identify hardware differences. The only hardware variance for the machines in the “different instance” experiment was that one Elasticsearch machine had a different CPU model. This can’t explain the variation in our OpenSearch results or variation among Elasticsearch instances with the same CPU.

The conclusion from this experiment is that we needed to control variation by running benchmarks on many different instances. Given the observed variation across different instances, more samples would make our results more consistent. We decided against using the same instance for all experiments because it was not feasible for the volume of experiments that needed to be run. We decided against using the same dedicated host because the variance was too high. This experiment on instance variation informed our final experimental design.

References

[1] Fisher, Ronald Aylmer. "Statistical methods for research workers." Breakthroughs in statistics: Methodology and distribution. New York, NY: Springer New York, 1970. 66–70.

[2] https://opensearch.org/docs/latest/search-plugins/knn/performance-tuning/#expert-level-build-vector-data-structures-on-demand

[3] https://www.edwardtufte.com/notebook/sparkline-theory-and-practice-edward-tufte/

[4] https://opensearch.org/docs/latest/benchmark/user-guide/concepts/#service-time

[5] https://github.com/trailofbits/opensearch-benchmark

[6] https://opensearch.org/benchmarks

[7] https://github.com/opensearch-project/OpenSearch/issues/17078

Appendix

Operations

Big5 (41 operations)

Date histogram:

- `composite-date_histogram-daily`: Aggregates daily content from 2022-12-30 to 2023-01-07.
- `date_histogram_hourly_agg`: Aggregates all content hourly.
- `date_histogram_minute_agg`: Aggregates all content by minute from 2023-01-01 to 2023-01-03.
- `range-auto-date-histo`: Aggregates `metrics.size` from [-infinity,-10], [-10,10], [10,100], [100,1000], [1000,2000], [2000,infinity], and by timestamps into 20 buckets.
- `range-auto-date-histo-with-metrics`: Aggregates `metrics.size` from [-infinity,100], [100,1000], [1000,2000], [2000,infinity], and retrieves the minimum, maximum, and average values for 10 buckets of timestamps.

Range queries:

- `range`: Searches for contents between 2023-01-01 and 2023-01-03.
- `keyword-in-range`: Matches "kernel" in `process.name` between 2023-01-01 and 2023-01-03.
- `range_field_conjunction_big_range_big_term_query`: Matches "systemd" in `process.name` with `metrics.size` between 1 and 100.
- `range_field_conjunction_small_range_big_term_query`: Matches `metrics.size` between 20 and 30.
- `range_field_conjunction_small_range_small_term_query`: Matches `aws.cloudwatch.log_stream` with "indigodagger" for `metrics.size` between 10 and 20.
- `range_field_disjunction_big_range_small_term_query`: Matches `aws.cloudwatch.log_stream` with "indigodagger" for `metrics.size` between 1 and 100.
- `range-agg-1`: Aggregates `metrics.size` over ranges [-infinity,-10], [-10,10], [10,100], [100,1000], [1000,2000], [2000,infinity].

- `range-agg-2`: Aggregates `metrics.size` over ranges [-infinity,100], [100,1000], [1000,2000], [2000,infinity].
- `range-numeric`: Matches range for `metrics.size` between 20 and 200.

Sorting:

- `asc_sort_timestamp`: Sorts contents by timestamp in ascending order.
- `asc_sort_timestamp_can_match_shortcut`: Sorts contents matching `process.name` to "kernel" by timestamp in ascending order.
- `asc_sort_timestamp_no_can_match_shortcut`: Sorts contents matching `process.name` to "kernel" by timestamp in ascending order. Uses a prefilter shard size of 100k.
- `asc_sort_with_after_timestamp`: Sorts contents by timestamp in ascending order after 2023-01-01T23:59:58.000Z.
- `desc_sort_timestamp`: Sorts contents by timestamp in descending order.
- `desc_sort_timestamp_can_match_shortcut`: Sorts contents matching process.name to "kernel" by timestamp in descending order.
- `desc_sort_timestamp_no_can_match_shortcut`: Sorts contents matching process.name to "kernel" by timestamp in descending order. Uses a prefilter shard size of 100k.
- `sort_keyword_can_match_shortcut`: Sorts contents matching process.name to "kernel" by `meta.file` in ascending order.
- `desc_sort_with_after_timestamp`: Sorts contents by timestamp in descending order after 2023-01-06T23:59:58.000Z.
- `sort_keyword_no_can_match_shortcut`: Sorts contents matching process.name to "kernel" by `meta.file` in descending order. Uses a prefilter shard size of 100k.
- `sort_numeric_asc`: Sorts contents by `metrics.size` in ascending order.
- `sort_numeric_asc_with_match`: Sorts contents matching `log.file.path` to "`/var/log/messages/solarshark`" by `metrics.size` in ascending order.
- `sort_numeric_desc`: Sorts contents by `metrics.size` in descending order.
- `sort_numeric_desc_with_match`: Sorts contents matching log.file.path to "`/var/log/messages/solarshark`" by `metrics.size` in descending order.

Term aggregations:

- `cardinality-agg-high`: Counts cardinality of `agent.name`.
- `cardinality-agg-low`: Counts cardinality of `cloud.region`.
- `composite_terms-keyword`: Matches contents with a timestamp between 2023-01-02T00:00:00 and 2023-01-02T10:00:00. Aggregates `process.name` in descending order, `cloud.region` in ascending order, and `aws.cloudwatch.log_stream` in ascending order.
- `composite-terms`: Matches contents with a timestamp between 2023-01-02T00:00:00 and 2023-01-02T10:00:00. Aggregates `process.name` in descending order and `cloud.region` in ascending order.
- `keyword-terms`: Aggregates contents by field `aws.cloudwatch.log_stream` in 500 buckets.
- `keyword-terms-low-cardinality`: Aggregates by field `aws.cloudwatch.log_stream` in 50 buckets.
- `multi_terms-keyword`: Aggregates by multiple terms `process.name` and `cloud.region` for contents where timestamp is between 2023-01-05T00:00:00 and 2023-01-05T05:00:00.

Text querying:

- `default`: Searches for all contents.
- `query-string-on-message`: Queries for the string "`message: monkey jackal bear`".
- `query-string-on-message-filtered`: Queries for the string "`message: monkey jackal bear`" between timestamps 2023-01-03T00:00:00 and 2023-01-03T10:00:00.
- `query-string-on-message-filtered-sorted-num`: Queries for the string "`message: monkey jackal bear`" between timestamps 2023-01-03T00:00:00 and 2023-01-03T10:00:00. Sorts by timestamp in ascending order.
- `scroll`: Searches for all contents, returning 25 pages where each page contains 1,000 results.
- `term`: Searches for log.file.path matching the value "`/var/log/messages/birdknight`".

NYC Taxis (7 operations)

Aggregation:

- `distance_amount_agg`: Aggregates by `trip_distance` and `total_amount` for trips where the distance is between 0 and 50.
- `date_histogram_agg`: Aggregates `dropoff_datetime` days for trips between 21/01/2015 and 01/01/2015.
- `autohisto_agg`: Aggregates `dropoff_datetime` days into 20 buckets for trips between 01/01/2015 and 21/01/2015.

Range queries:

- `range`: Searches for `total_amount` values between 5 and 15.

Sorting:

- `desc_sort_tip_amount`: Returns all contents by `tip_amount` in descending order.
- `asc_sort_tip_amount`: Returns all contents by `tip_amount` in ascending order.

Text Querying:

- `default`: Searches for all contents.

PMC (6 operations)

Aggregation:

- `articles_monthly_agg_uncached`: Aggregates articles by month without a cache.
- `articles_monthly_agg_cached`: Aggregates articles by month with a cache.

Text querying:

- `default`: Searches for all contents.
- `phrase`: Searches for the term “newspaper coverage” in the body.
- `scroll`: Searches for all contents, returning 25 pages where each page contains 100 results.
- `term`: Searches for the term “physician” in the body.

NOAA (24 operations)

Date histogram:

- `date-histo-entire-range`: Aggregates all contents by 2,000 day intervals.

- `date-histo-geohash-grid`: Returns geocoordinates from C to 9. Aggregates dates by one week and `station.location`. by `geohash_grid`.
- `date-histo-geotile-grid`: Returns geocoordinates from C to 9. Aggregates dates by one week and `station.location` by `geotile_grid`.
- `date-histo-histo`: Aggregates dates by one week, and `TAVG` by 10.
- `date-histo-numeric-terms`: Aggregates dates and `TAVG` by one week.
- `date-histo-string-significant-terms-via-default-strategy`: Aggregates dates and `station.country` by one week.
- `date-histo-string-significant-terms-via-global-ords`: Aggregates dates and `station.country` and `global_ordinals` by one week.
- `date-histo-string-significant-terms-via-map`: Aggregates dates and `station.country` and map by one week.
- `date-histo-string-terms-via-default-strategy`: Aggregates dates and `station.country` by one week.
- `date-histo-string-terms-via-global-ords`: Aggregates dates and `station.country` by one week.
- `date-histo-string-terms-via-map`: Aggregates dates and `station.country` by one week.

Range and date histogram:

- `range-auto-date-histo`: Aggregates results from `TMAX` ranges [-infinity,-10], [-10,0], [0,10], [10,20], [20,30], [30,infinity] into 20 buckets by date.
- `range-auto-date-histo-with-metrics`: Aggregates results from `TMAX` ranges [-infinity,-10], [-10,0], [0,10], [10,20], [20,30], [30,infinity] into 20 buckets by date, with the minimum of `TMIN`, average of `TAVG`, and maximum of `TMAX`.
- `range-auto-date-histo-with-time-zone`: Aggregates results from `TMAX` ranges [-infinity,-10], [-10,0], [0,10], [10,20], [20,30], [30,infinity] into 20 buckets by date in the New York time zone.
- `range-date-histo`: Aggregates results from `TMAX` ranges [-infinity,-10], [-10,0], [0,10], [10,20], [20,30], [30,infinity] into one-week intervals.
- `range-date-histo-with-metrics`: Aggregates results from `TMAX` ranges [-infinity,-10], [-10,0], [0,10], [10,20], [20,30], [30,infinity] into one-week intervals, with the minimum of `TMIN`, average of `TAVG`, and maximum of `TMAX`.

Range queries:

- `range-aggregation`: Aggregates results from `TMAX` ranges [-infinity,-10], [-10,0], [0,10], [10,20], [20,30], [30,infinity].
- `range-numeric-significant-terms`: Aggregates results from `TMAX` ranges [-infinity,-10], [-10,0], [0,10], [10,20], [20,30], [30,infinity] by date.

Term aggregations:

- `keyword-terms`: Aggregates contents by `station.id` in 500 buckets.
- `keyword-terms-low-cardinality`: Aggregates contents by `station.country` in 200 buckets.
- `keyword-terms-low-cardinality-min`: Aggregates `TMIN` by `station.id` in 200 buckets.
- `keyword-terms-min`: Aggregates TMIN by `station.id` in 500 buckets.
- `keyword-terms-numeric-terms`: Aggregates `TMAX` by date and `station.id` in 500 buckets.
- `numeric-terms-numeric-terms`: Aggregates `TAVG` by `TMAX` in 100 buckets.

Vectorsearch (1 operation)

ML:

- `prod-queries`: Performs approximate k-NN search.

NOAA-Semantic-Search (20 operations)

Note: All operations use `nlp-min-max-arithmetic-search-pipeline`. This is a search phase results processor that runs between query and fetch time. It normalizes scores with min-max normalization and combines scores using an arithmetic mean.

Aggregation:

- `aggs-query-date-histo-geohash-grid-hybrid`: Performs a hybrid query of the following queries: term query where `station.country_code` is "JA", range query where 0 <= `TRANGE` <= 30, and range query where `date` >= "2016-06-04". Performs a date aggregation on field `date` with a calendar interval of "1M" and a `geohash_grid` aggregation on `station.location`.
- `aggs-query-date-histo-geohash-grid-hybrid-one-subquery`: Performs a hybrid query using a range query where -100 <= `TRANGE` <= -50. Performs a date

aggregation on field `date` with a calendar interval of "1M" and a `geohash_grid` aggregation on `station.location`.

- `aggs-query-date-histo-geohash-grid-hybrid-one-subquery-large-subset`: Performs a hybrid query using a range query where 1 <= `TRANGE` <= 35. Performs a date aggregation on field `date` with a calendar interval of "1M" and a `geohash_grid` aggregation on `station.location`.
- `aggs-query-date-histo-geohash-grid-hybrid-one-subquery-medium-subset`: Performs a hybrid query using a range query where -90 <= `TRANGE` <= -7. Performs a date aggregation on field `date` with a calendar interval of "1M" and a `geohash_grid` aggregation on `station.location`.
- `aggs-query-min-avg-sum-hybrid`: Performs a hybrid query of the following queries: term query where `station.country_code` is "JA", range query where 0 <= `TRANGE` <= 30, and range query where `date` >= "2016-06-04". Performs the aggregations: max of `TMAX`, average of `TAVG`, and sum of `THIC`.
- `aggs-query-min-avg-sum-hybrid-one-subquery`: Performs a hybrid query using a range query where -100 <= `TRANGE` <= -50. Performs the aggregations: max of `TMAX`, average of `TAVG`, and sum of `THIC`.
- `aggs-query-min-avg-sum-hybrid-one-subquery-large-subset`: Performs a hybrid query using a range query where 1 <= `TRANGE` <= 35. Performs the aggregations: max of `TMAX`, average of `TAVG`, and sum of `THIC`.
- `aggs-query-min-avg-sum-hybrid-one-subquery-medium-subset`: Performs a hybrid query using a range query where -90 <= `TRANGE` <= -7. Performs the aggregations: max of `TMAX`, average of `TAVG`, and sum of `THIC`.
- `aggs-query-range-numeric-significant-terms-hybrid`: Performs a hybrid query of the following queries: term query where `station.country_code` is "JA", range query where 0 <= `TRANGE` <= 30, and range query where `date` >= "2016-06-04". Performs the aggregations: range aggregation on `TMAX` over [-infinity, -10], [-10, 0], [0, 10], [10, 20], [20, 30], and [30, infinity] and a significant terms aggregation on `date`.
- `aggs-query-range-numeric-significant-terms-hybrid-one-subquery`: Performs a hybrid query using a range query where -100 <= `TRANGE` <= -50. Performs the aggregations: range aggregation on `TMAX` over [-infinity, -10], [-10, 0], [0, 10], [10, 20], [20, 30], and [30, infinity] and a significant terms aggregation on `date`.
- `aggs-query-range-numeric-significant-terms-hybrid-one-subquery-large-subset`: Performs a hybrid query using a range query where 1 <= `TRANGE` <= 35. Performs the aggregations: range aggregation on `TMAX` over [-infinity, -10], [-10,

0], [0, 10], [10, 20], [20, 30], and [30, infinity] and a significant terms aggregation on `date`.

- `aggs-query-range-numeric-significant-terms-hybrid-one-subquery-medium-subset`: Performs a hybrid query using a range query where -90 <= `TRANGE` <= -7. Performs the aggregations: range aggregation on `TMAX` over [-infinity, -10], [-10, 0], [0, 10], [10, 20], [20, 30], and [30, infinity] and a significant terms aggregation on `date`.
- `aggs-query-term-min-hybrid`: Performs a hybrid query of the following queries: term query where `station.country_code` is "JA", range query where 0 <= `TRANGE` <= 30, and range query where `date` >= "2016-06-04". Performs the aggregations: terms aggregation on `station.id` with 500 buckets and a min aggregation on `TMIN`.
- `aggs-query-term-min-hybrid-one-subquery`: Performs a hybrid query using a range query where -100 <= `TRANGE` <= -50. Performs the aggregations: terms aggregation on `station.id` with 500 buckets and a min aggregation on `TMIN`.
- `aggs-query-term-min-hybrid-one-subquery-large-subset`: Performs a hybrid query using a range query where 1 <= `TRANGE` <= 35. Performs the aggregations: terms aggregation on `station.id` with 500 buckets and a min aggregation on `TMIN`.
- `aggs-query-term-min-hybrid-one-subquery-medium-subset`: Performs a hybrid query using a range query where -90 <= `TRANGE` <= -7. Performs the aggregations: terms aggregation on `station.id` with 500 buckets and a min aggregation on `TMIN`.

Hybrid query:

- `hybrid-query-only-range`: Performs a hybrid query using a range query where -100 <= `TRANGE` <= -50.
- `hybrid-query-only-range-large-subset`: Performs a hybrid query using a range query where 1 <= `TRANGE` <= 35.
- `hybrid-query-only-range-medium-subset`: Performs a hybrid query using a range query where -90 <= `TRANGE` <= -7.
- `hybrid-query-only-term-range-date`: Performs a hybrid query of the following queries: term query where `station.country_code` is "JA", range query where 0 <= `TRANGE` <= 30, and range query where `date` >= "2016-06-04".

OpenSearch v2.18 - Performance Results

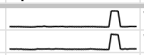
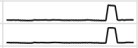
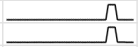
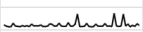
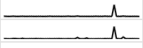
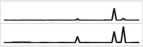
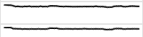
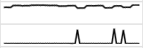
In this section, we highlight performance measurements for OpenSearch for the latest versions of OpenSearch (2.18.0) and Elasticsearch (8.15.4). The following results compare the workloads: Big5, NYC Taxis, PMC, and NOAA.

To measure statistical certainty, we calculated the p-value and statistical power of each operation in each workload for each engine version. To do this, we looked at *all* reported values for each execution of each run (instead of just the 90th percentile value determined by OSB) to ensure a holistic view of statistical significance. For p-values less than 0.05 (indicating a statistically significant difference in performance), OpenSearch outperformed Elasticsearch on 25 of 78 tasks (32%) using geometric mean. Elasticsearch outperformed OpenSearch on 53 of 78 tasks (68%). We discuss statistical certainty more in the Performance Validation section.

Notably, while OpenSearch is slower on most tasks, the tasks it is faster on are 1–3 orders of magnitude faster than Elasticsearch.

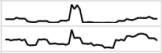
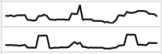
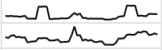
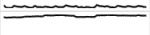
Sparklines

We zoom in on the most recent version of OpenSearch (2.18.0) and Elasticsearch (8.15.4). We plot sparklines using the *same* scale for each operation (noting the maximum and minimum value for each plot) to illustrate how service times compare between each engine. Note that a smaller value is better (i.e., faster). We do not compare NOAA Semantic Search because Elasticsearch does not currently support that workload.

Workload	Category	Operation	OpenSearch 2.18.0	Elasticsearch 8.15.4	Min	Max
big5	Sorting	asc_sort_timestamp			4.62	23.88
big5	Sorting	asc_sort_timestamp_can_match_shortcut			4.81	20.37
big5	Sorting	asc_sort_timestamp_no_can_match_shortcut			4.84	20.21
big5	Sorting	asc_sort_with_after_timestamp			177.98	469.55
big5	Term Aggregations	cardinality-agg-high			702.33	2,559.96
big5	Term Aggregations	cardinality-agg-low			0.89	2.43
big5	Date Histogram	composite-date_histogram-daily			1.00	1,813.50
big5	Term Aggregations	composite-terms			214.22	246.86
big5	Term Aggregations	composite_terms-keyword			309.13	376.26
big5	Date Histogram	date_histogram_hourly_agg			4.52	1,526.28
big5	Date Histogram	date_histogram_minute_agg			32.00	1,333.46
big5	General Operations	default			1.18	2.60
big5	Sorting	desc_sort_timestamp			3.05	66.00
big5	Sorting	desc_sort_timestamp_can_match_shortcut			2.61	13.93
big5	Sorting	desc_sort_timestamp_no_can_match_shortcut			2.60	14.02
big5	Sorting	desc_sort_with_after_timestamp			64.79	341.58
big5	Range Queries	keyword-in-range			12.15	81.13
big5	Term Aggregations	keyword-terms			22.60	3,359.73
big5	Term Aggregations	keyword-terms-low-cardinality			19.12	3,315.97
big5	Term Aggregations	multi_terms-keyword			725.21	1,394.02
big5	Text Querying	query-string-on-message			3.29	158.43
big5	Text Querying	query-string-on-message-filtered			9.25	29.19
big5	Text Querying	query-string-on-message-filtered-sorted-num			23.55	27.82
big5	Range Queries	range			6.44	9.95
big5	Range Queries	range-agg-1			0.88	1.52
big5	Range Queries	range-agg-2			0.87	1.32
big5	Date Histogram	range-auto-date-histo			2,149.37	10,449.84
big5	Date Histogram	range-auto-date-histo-with-metrics			4,978.15	22,022.72
big5	Range Queries	range-numeric			0.64	1.00
big5	Range Queries	range_field_conjunction_big_range_big_term_query			0.74	0.97
big5	Range Queries	range_field_conjunction_small_range_big_term_query			0.65	0.99
big5	Range Queries	range_field_conjunction_small_range_small_term_query			0.72	0.96
big5	Range Queries	range_field_disjunction_big_range_small_term_query			0.73	1.01
big5	General Operations	scroll			204.15	405.81
big5	Sorting	sort_keyword_can_match_shortcut			1.94	2.37
big5	Sorting	sort_keyword_no_can_match_shortcut			1.95	2.42
big5	Sorting	sort_numeric_asc			2.50	10.32
big5	Sorting	sort_numeric_asc_with_match			0.67	1.00
big5	Sorting	sort_numeric_desc			3.10	9.67
big5	Sorting	sort_numeric_desc_with_match			0.65	0.97
big5	Text Querying	term			0.72	1.04

Workload	Category	Operation	OpenSearch 2.18.0	Elasticsearch 8.15.4	Min	Max
nyc_taxis	Sorting	asc_sort_tip_amount			2.69	6,515.93
nyc_taxis	Aggregation	autohisto_agg			3.06	304.68
nyc_taxis	Aggregation	date_histogram_agg			3.02	5.64
nyc_taxis	General Operations	default			1.13	3.61
nyc_taxis	Sorting	desc_sort_tip_amount			3.19	6,465.69
nyc_taxis	Aggregation	distance_amount_agg			2,500.91	11,764.06
nyc_taxis	Range Queries	range			55.50	218.17

Workload	Category	Operation	OpenSearch 2.18.0	Elasticsearch 8.15.4	Min	Max
pmc	Aggregation	articles_monthly_agg_cached			0.95	1.80
pmc	Aggregation	articles_monthly_agg_uncached			5.70	7.55
pmc	General Operations	default			2.24	5.14
pmc	Text Querying	phrase			2.92	5.31
pmc	General Operations	scroll			221.55	975.67
pmc	Text Querying	term			2.55	5.66

Workload	Category	Operation	OpenSearch 2.18.0	Elasticsearch 8.15.4
noaa	Date Histogram	date-histo-entire-range		
noaa	Date Histogram	date-histo-geohash-grid		
noaa	Date Histogram	date-histo-geotile-grid		
noaa	Date Histogram	date-histo-histo		
noaa	Date Histogram	date-histo-numeric-terms		
noaa	Date Histogram	date-histo-string-significant-terms-via-default-strategy		
noaa	Date Histogram	date-histo-string-significant-terms-via-global-ords		
noaa	Date Histogram	date-histo-string-significant-terms-via-map		
noaa	Date Histogram	date-histo-string-terms-via-default-strategy		
noaa	Date Histogram	date-histo-string-terms-via-global-ords		
noaa	Date Histogram	date-histo-string-terms-via-map		
noaa	Term Aggregations	keyword-terms		
noaa	Term Aggregations	keyword-terms-low-cardinality		
noaa	Term Aggregations	keyword-terms-low-cardinality-min		
noaa	Term Aggregations	keyword-terms-min		
noaa	Term Aggregations	keyword-terms-numeric-terms		
noaa	Term Aggregations	numeric-terms-numeric-terms		
noaa	Range Queries	range-aggregation		
noaa	Range & Date Histogram	range-auto-date-histo		
noaa	Range & Date Histogram	range-auto-date-histo-with-metrics		
noaa	Range & Date Histogram	range-auto-date-histo-with-time-zone		
noaa	Range & Date Histogram	range-date-histo		
noaa	Range & Date Histogram	range-date-histo-with-metrics		
noaa	Range Queries	range-numeric-significant-terms		

Categorized Results

We zoom in on those operations that at least doubled the performance (i.e., took less than half as long to complete) over each engine on average. When categorizing each operation, OpenSearch outperforms Elasticsearch with ascending sort and keyword term aggregation queries by more than 2x. Elasticsearch outperforms OpenSearch with descending sort and date histogram queries by more than 2x.

Specific service time values in this section can be found in the Appendix.

Sort

The goal of Sorting operations is to arrange values in either ascending or descending order based on specific fields.

- OpenSearch is **2.2x–4.6x faster** on Big5 ascending sort operations (`asc_sort_timestamp`, `asc_sort_timestamp_can_match_shortcut`, `asc_sort_timestamp_no_can_match_shortcut`, `asc_sort_with_after_timestamp`) and **1,700x–2,000x faster** on NYC Taxis sorting operations (`asc_sort_tip_amount`, `desc_sort_tip_amount`).
- Elasticsearch is **1.4x–2.8x faster** on Big5 descending sort operations (`desc_sort_timestamp`, `desc_sort_timestamp_can_match_shortcut`, `desc_sort_timestamp_no_can_match_shortcut`, `desc_sort_with_after_timestamp`).

Date histograms

Date histograms group documents based on date or timestamp fields into buckets according to a specified time interval. They are particularly useful for time-series analysis and visualizing data trends over time.

- OpenSearch is **37x–1,500x faster** on Big5 date histogram operations (`composite-date_histogram-daily`, `date_histogram_hourly_agg`, `date_histogram_minute_agg`).
- Elasticsearch is **3.7x faster** on Big5 date histogram operations (`range-auto-date-histo`, `range-auto-date-histo-with-metrics`) and **4.1x–4.9x faster** on NOAA operations (`date-histo-geohash-grid`, `date-histo-geotile-grid`, `date-histo-histo`, `date-histo-histo`).
- Elasticsearch is **5.7x–9.5x faster** on NOAA range and date histogram operations (`range-auto-date-histo`, `range-auto-date-histo-with-metrics`, `range-auto-date-histo-with-time-zone`, `range-date-histo`, `range-date-histo-with-metrics`).

Term Aggregations

Term Aggregations group documents based on specific field values. They create buckets where each bucket represents a unique term in the specified field, and documents are grouped into these buckets accordingly.

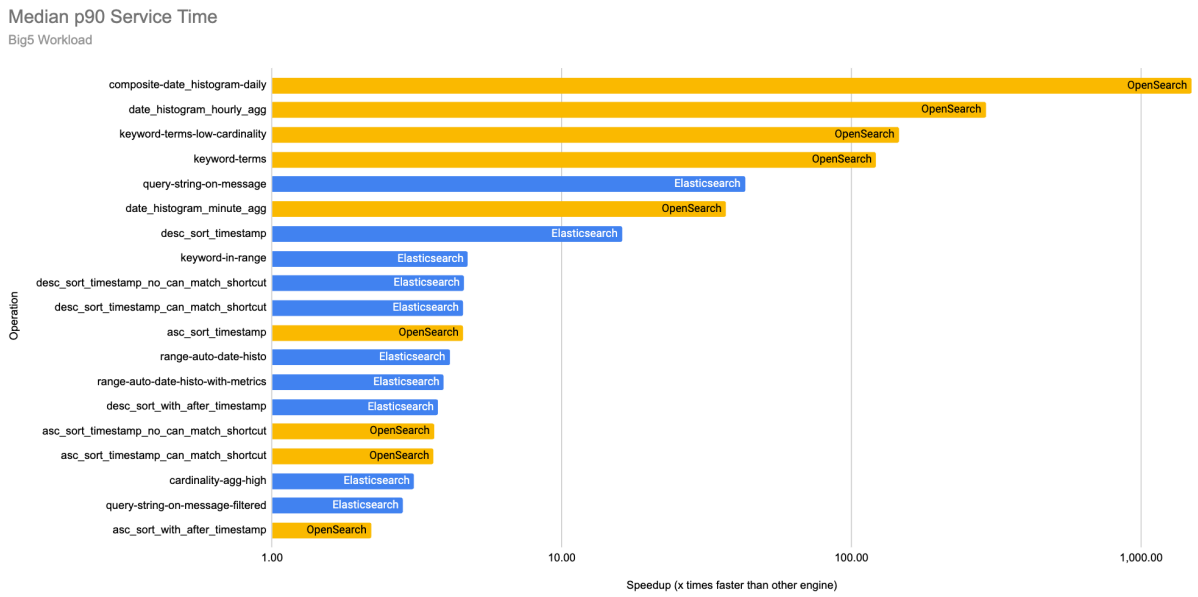
- OpenSearch is **122x-146x faster** on Big5 term aggregation operations (keyword-terms, keyword-terms-low-cardinality) and **3.4x–7x faster** on NOAA term aggregation operations (`keyword-terms`, `keyword-terms-low-cardinality`).
- Elasticsearch is **3.1x faster** on Big5 term aggregation operation (`cardinality-agg-high`) and **7.8x faster** on NOAA operation (`keyword-terms-low-cardinality-min`).

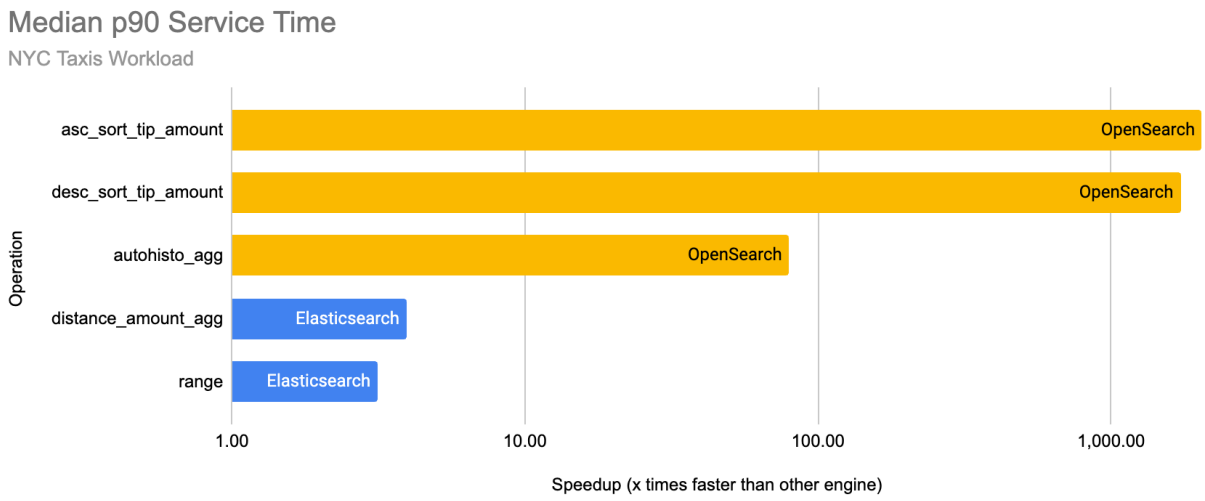
Others

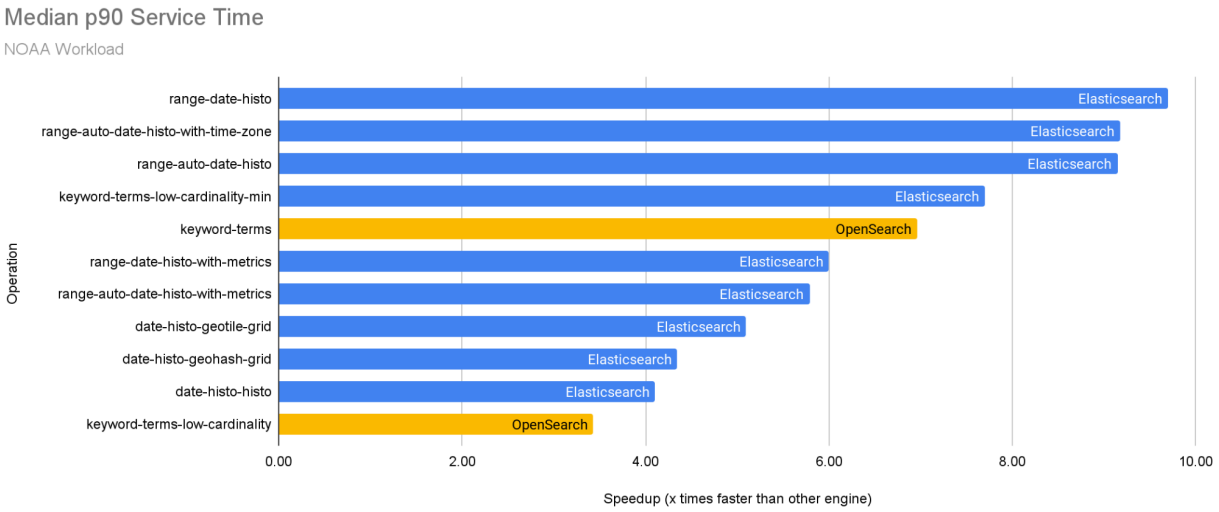
Other notable operation comparisons include:

- OpenSearch is **80x faster** on a NYC Taxis aggregation operation (`autohisto_agg`).
- Elasticsearch is **2.7x–38x faster** on Big5 text querying operations (`query-string-on-message`, `query-string-on-message-filtered`).
- Elasticsearch is **3.1x faster** on NYC Taxis range operation (`range`).
- Elasticsearch is **4.2x faster** on NYC Taxis aggregation operation (`distance_amount_agg`).

To illustrate these results, the following graph shows where the y-axis represents an operation and the x-axis represents how many times faster an engine is over the other. From these graphs we can see that when OpenSearch is faster, it is significantly faster than Elasticsearch, while Elasticsearch outperforms OpenSearch on the NOAA workload, though by a smaller degree.







Finally, only one PMC workload operation `scroll` had a speedup of over 2x. Elasticsearch is **3.7x faster** than OpenSearch.

OpenSearch v2.18 - Vectorsearch

This section reports results from our vectorsearch workload (separately from the other four workloads) because of the distinct nature of this workload and the changes we made to its configuration.

OpenSearch supports three vectorsearch engines: NMSLIB, FAISS, and Lucene. Elasticsearch supports only Lucene. Any reported values in the charts below for Elasticsearch are when it is run with Lucene. For brevity, we specify a search engine and the vectorsearch engine used in the format: search engine (vector engine).

Only one task was tested in the vectorsearch workload: `prod-queries`.

Standard Configuration

The standard configuration refers to a configuration described in the Evaluation Setup that we used for our nightly benchmarks. We tested Vectorsearch with the standard configuration from December 5, 2024, to December 16, 2024.

Comparing each OpenSearch (2.16.0, 2.17.1, 2.18.0) vector engine against Elasticsearch (Lucene) 8.15.4 across OpenSearch versions yielded the following findings:

- OpenSearch (FAISS) was **13.8%–15.6% faster**
- OpenSearch (NMSLIB) was **11.3%–13.0% faster**
- OpenSearch (Lucene) was **164.3%–258.2% slower**

The default vector engine for OpenSearch is FAISS for >=2.18.0 and NMSLIB for <2.18.0. We compared each OpenSearch vector engine against Elasticsearch (Lucene) (instead of only

the default vector engine) so that we could understand how each available OpenSearch configuration would perform. We also compared the OpenSearch vector engines against each other to make the performance differences between them more explicit. Averaging the results of each version yielded the following results:

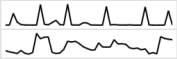
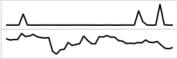
- OpenSearch (FAISS) averaged **7.48 ms**
- OpenSearch (NMSLIB) averaged **7.70 ms**
- OpenSearch (Lucene) averaged **28.55 ms**

The slower performance of OpenSearch (Lucene) was driven by slow outlier runs. Although the average was 28.55 ms, the median value was 16.54 ms. The slowest outlier was 252.34 ms. As we discuss in the following Additional Configuration Experiments section, we are able to reduce outliers in OpenSearch (Lucene) in an experiment with a modified JVM Heap size and snapshot process.

Sparklines

Similar to the above section we created two sets of sparkline plots to visualize all of the data for p90 service times. Lower values are better (i.e., faster).

First, we show the *variation* in service times (i.e., not using the same scale) for each engine.

Workload	Category	Operation	Workload Subtype	OpenSearch 2.16.0	OpenSearch 2.17.1	OpenSearch 2.18.0	Elasticsearch 8.15.4
vectorsearch	ML	prod-queries	faiss-cohere-10m				
vectorsearch	ML	prod-queries	lucene-cohere-10m				
vectorsearch	ML	prod-queries	nmslib-cohere-10m				

Next, we show the *same scale* for service times, comparing the latest version of OpenSearch (2.18.0) and Elasticsearch (8.15.4). The maximum and minimum values of the sparklines are displayed to the right of each plot.

Workload	Category	Operation	Workload Subtype	OpenSearch 2.18.0	Elasticsearch 8.15.4	Min	Max
vectorsearch	ML	prod-queries	faiss-cohere-10m			7.23	9.17
vectorsearch	ML	prod-queries	lucene-cohere-10m			8.39	128.78
vectorsearch	ML	prod-queries	nmslib-cohere-10m			7.16	9.17

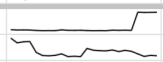
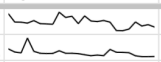
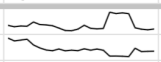
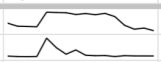
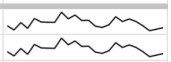
Additional Configuration Experiments

We performed additional experiments to explore other vectorsearch configuration scenarios.

Sparklines: Force Merge Enabled

Similar to the above section we created two sets of sparkline plots to visualize all of the data for p90 service times. Lower values are better (i.e., faster).

First, we show the *variation* in service times (i.e., not using the same scale) for each engine.

Workload	Category	Operation	Workload Subtype	OpenSearch 2.16.0	OpenSearch 2.17.1	OpenSearch 2.18.0	OpenSearch 2.19.0	Elasticsearch 8.15.4
vectorsearch	ML	prod-queries	faiss-cohere-10m					
vectorsearch	ML	prod-queries	lucene-cohere-10m					
vectorsearch	ML	prod-queries	nmslib-cohere-10m					

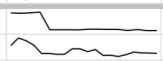
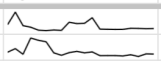
Next, we show the *same scale* for service times, comparing the latest versions of OpenSearch (2.18.0 and 2.19-development) and Elasticsearch (8.15.4). The maximum and minimum values of the sparklines are displayed to the right of each plot.

Workload	Category	Operation	Workload Subtype	OpenSearch 2.18.0	OpenSearch 2.19.0	Elasticsearch 8.15.4	Min	Max
vectorsearch	ML	prod-queries	faiss-cohere-10m				7.10	8.85
vectorsearch	ML	prod-queries	lucene-cohere-10m				8.43	17.80
vectorsearch	ML	prod-queries	nmslib-cohere-10m				7.19	8.85

Sparklines: Force Merge Disabled

These sparklines were created similarly, except for measuring performance when force merge was disabled.

First, we show the *variation* in service times (i.e., not using the same scale) for each engine.

Workload	Category	Operation	Workload Subtype	OpenSearch 2.16.0	OpenSearch 2.17.1	OpenSearch 2.18.0	OpenSearch 2.19.0	Elasticsearch 8.15.4
vectorsearch	ML	prod-queries	faiss-cohere-10m					
vectorsearch	ML	prod-queries	lucene-cohere-10m					
vectorsearch	ML	prod-queries	nmslib-cohere-10m					

Next, we show the *same scale* for service times, comparing the latest versions of OpenSearch and Elasticsearch.

Workload	Category	Operation	Workload Subtype	OpenSearch 2.18.0	OpenSearch 2.19.0	Elasticsearch 8.15.4	Min	Max
vectorsearch	ML	prod-queries	faiss-cohere-10m				12.45	67.37
vectorsearch	ML	prod-queries	lucene-cohere-10m				12.45	83.40
vectorsearch	ML	prod-queries	nmslib-cohere-10m				12.45	35.51

OpenSearch vs. Elasticsearch on Big5 Across Versions

For brevity, OS means OpenSearch, and ES means Elasticsearch. The values are the median of the 90th percentile values of service time (ST) in milliseconds.

Ratio cell colors:

- Light red represents Elasticsearch being between 0 and 2x faster than OpenSearch
- Dark red represents Elasticsearch being more than 2x faster than OpenSearch
- Light green represents OpenSearch being between 0 and 2x faster than Elasticsearch
- Dark green represents OpenSearch being more than 2x faster than Elasticsearch

Operation	ES 8.15.4 P90 ST	OS 2.16.0 P90 ST	OS 2.17.1 P90 ST	OS 2.18.0 P90 ST	Ratio ES 8.15.4 /	Ratio ES 8.15.4 /	Ratio ES 8.15.4 /

					OS 2.16.0	OS 2.17.1	OS 2.18.0
composite-date_histogram-daily	1,685.86	1.08	1.13	1.13	1,567.03	1,494.42	1,494.28
date_histogram_hourly_agg	1,435.51	4.66	4.77	4.93	308.34	301.17	291.00
date_histogram_minute_agg	1,260.78	32.65	33.16	34.14	38.61	38.02	36.93
range-auto-date-histo	2,314.83	8,905.45	8,661.77	9,500.39	0.26	0.27	0.24
range-auto-date-histo-with-metrics	5,311.37	20,052.59	19,588.54	20,823.50	0.26	0.27	0.26
range	8.73	8.25	8.26	7.17	1.06	1.06	1.22
keyword-in-range	12.79	12.00	12.20	60.76	1.07	1.05	0.21
range_field_conjunction_big_range_big_term_query	0.80	0.81	0.85	0.82	0.99	0.94	0.97
range_field_conjunction_small_range_big_term_query	0.69	0.72	0.77	0.75	0.96	0.90	0.93
range_field_conjunction_small_range_small_term_query	0.78	0.80	0.82	0.81	0.98	0.95	0.96
range_field_disjunction_big_range_small_term_query	0.78	0.79	0.82	0.80	0.99	0.95	0.98
range-agg-1	1.27	0.95	1.00	0.99	1.34	1.28	1.29
range-agg-2	1.14	0.93	0.97	0.99	1.23	1.17	1.15
range-numeric	0.68	0.70	0.73	0.74	0.97	0.93	0.92
asc_sort_timestamp	22.99	4.90	4.94	5.03	4.70	4.66	4.57
asc_sort_timestamp_can_match_shortcut	18.88	4.95	5.01	5.22	3.81	3.77	3.62

asc_sort_timestamp_no_can_match_shortcut	18.86	4.95	5.01	5.20	3.81	3.76	3.63
asc_sort_with_after_timestamp	408.96	178.21	190.11	186.21	2.29	2.15	2.20
desc_sort_timestamp	3.16	4.51	4.67	51.09	0.70	0.68	0.06
desc_sort_timestamp_can_match_shortcut	2.69	4.00	3.65	12.30	0.67	0.74	0.22
desc_sort_timestamp_no_can_match_shortcut	2.68	3.99	3.66	12.34	0.67	0.73	0.22
sort_keyword_can_match_shortcut	2.19	1.84	2.16	2.16	1.19	1.01	1.02
desc_sort_with_after_timestamp	70.54	342.20	194.76	264.33	0.21	0.36	0.27
sort_keyword_no_can_match_shortcut	2.20	1.84	2.18	2.17	1.20	1.01	1.01
sort_numeric_asc	2.57	10.08	7.23	3.76	0.26	0.36	0.68
sort_numeric_asc_with_match	0.71	0.76	0.80	0.79	0.94	0.89	0.90
sort_numeric_desc	3.21	9.46	8.26	4.38	0.34	0.39	0.73
sort_numeric_desc_with_match	0.71	0.76	0.80	0.78	0.93	0.88	0.90
cardinality-agg-high	765.29	2,476.57	2,402.41	2,369.55	0.31	0.32	0.32
cardinality-agg-low	0.99	1.75	1.76	1.78	0.56	0.56	0.55
composite_ter	330.24	356.91	348.57	341.81	0.93	0.95	0.97

ms-keyword							
composite-terms	228.39	230.71	234.36	228.64	0.99	0.97	1.00
keyword-terms	3,022.38	25.42	25.09	24.80	118.88	120.44	121.86
keyword-terms-low-cardinality	3,007.98	21.00	21.10	20.60	143.23	142.53	145.99
multi_terms-keyword	1,361.28	775.81	763.69	737.92	1.75	1.78	1.84
query-string-on-message	3.38	130.50	129.68	145.00	0.03	0.03	0.02
query-string-on-message-filtered	9.54	25.32	25.66	27.03	0.38	0.37	0.35
query-string-on-message-filtered-sorted-num	25.47	23.60	24.39	25.73	1.08	1.04	0.99
term	0.80	0.79	0.83	0.81	1.01	0.96	0.98
default	1.25	2.22	2.23	2.30	0.56	0.56	0.54
scroll	213.05	237.95	235.59	371.95	0.90	0.90	0.57