Query Planning for Range Queries in Elasticsearch

If you are a frequent reader of this blog, you probably know that a lot of effort has already been put into making range queries faster. This time we are going to talk about recent improvements to the specific yet common case of range queries when they are used in conjunctions, ie. when they are ANDed with other queries.

Why are conjunctions different?

The way conjunctions work is by iterating over matches from the most selective clause and verifying whether other required clauses match too. This means that we need two operations to be fast in order for conjunctions to be fast as well:

- iterating over all matches,
- verifying whether a particular document matches.

Ranges have been problematic so far because they could only iterate over all matches efficiently, not verify individual documents. Numerics are indexed using a tree structure, called "points", that is organized by **value**: it can help you find matching documents for a given range of value, but if you want to verify whether a particular document matches, you have no choice but to compute the set of matching documents and test whether it contains your document. Said otherwise, even if you only need to verify a few documents, you need to evaluate all documents that match the range!

There must be a better way

Good news is that in a majority of cases, Elasticsearch also has doc

values enabled on numeric fields. Doc values are a per-field lookup structure that associates each document with the set of values it contains. This is great for sorting or aggregations, but we could also use them in order to verify matches: this is the right data-structure for it.

Should we just use doc values all the time to execute ranges? No. Doc values are a form of columnar storage, not an indexed structure. If you want to evaluate all documents that match a range with doc values, there is no other choice but to perform a linear scan and verify every document, which is slow.

In summary here is what a better query plan for range queries would look like:

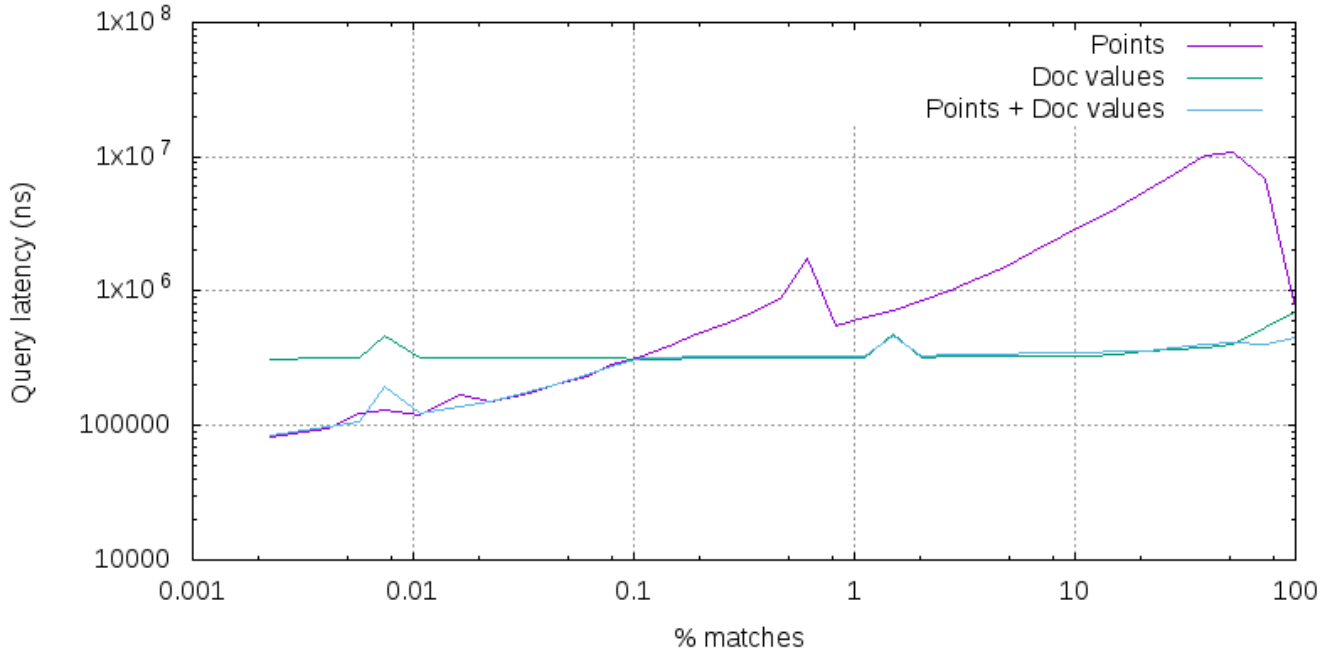
- iterate over all matches? → use the index
- verify whether a particular document matches? → use doc values

As a consequence, we introduced a new mechanism that allows queries to know whether they will be used for sequential access (iterating over all matches) or random access (verifying that some documents match) as well as a query wrapper called `IndexOrDocValuesQuery` that wraps a query that is good at iterating over matches and a query that is good at verifying matches. Then `IndexOrDocValuesQuery` delegates to the appropriate one depending on how it is used. This query wrapper will be used transparently by Elasticsearch on numeric fields that are both indexed and have doc values, which is the default.

Something that is interesting to notice here is that this query planning optimization does not only depend on the fields that are used and their cardinalities, it goes further and estimates the total number of matches for each node of the query tree in order to make good decisions. This means that taking a query and slightly changing the range of values might completely change how the query is executed under the hood.

Benchmarks

Obviously all this can't be perfect. It relies on the fact that the cost of iterating over all matches or verifying individual documents is the same for all queries, so while the theory fits nicely, it could be that practical results are disappointing. Let's see how this works in practice:

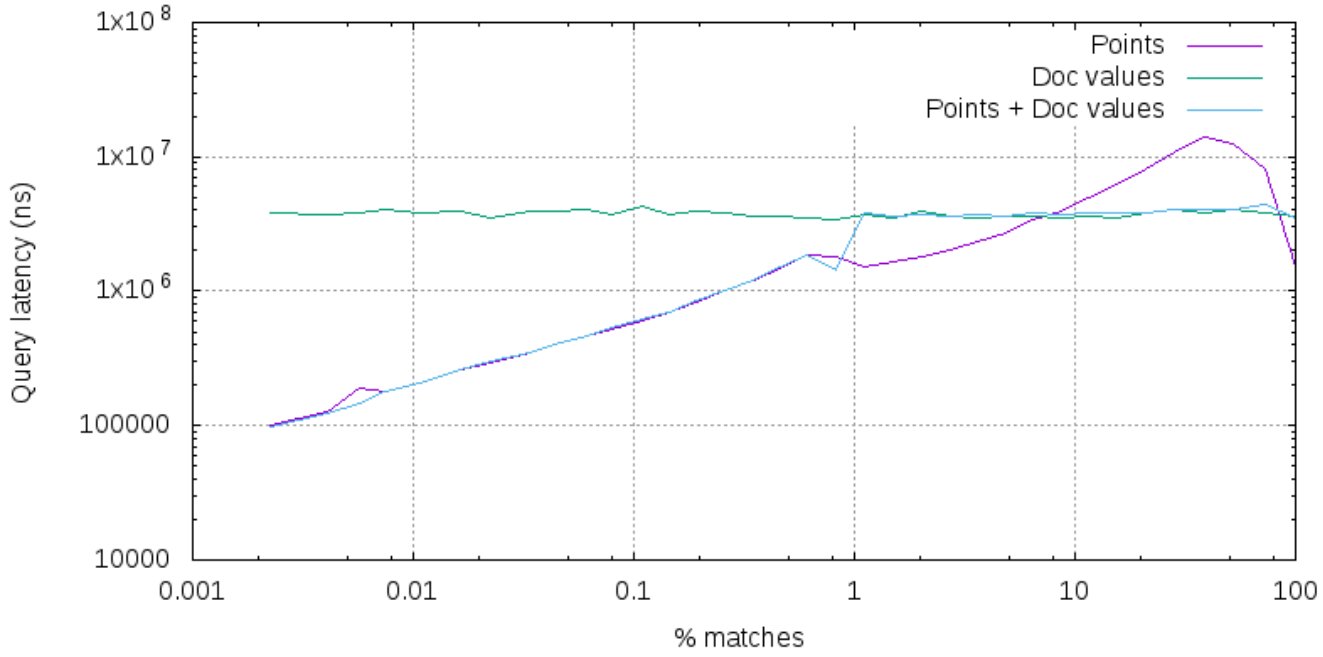


Query latency for 0.1% term and range

For this benchmark, I took a 10M documents subset of Wikipedia with a body field that contains the text of the article and a numeric field that stores the last time the article was updated. A query that has a term query that matches 10,000 documents (0.1% of the index) is intersected with ranges that match various numbers of documents. On the X axis is the number of documents that the range matches, and on the Y axis is the time it took to run the query. Then I performed 3 measurements: once using an index structure all the time ("Points"), once using doc values all the time ("Doc values") and once using the new `IndexOrDocValuesQuery` ("Points + Doc values"). When reading this chart, beware that it uses a logarithmic scale both on the X and on the Y axis. It might look a bit flat due to the logarithmic scale but the performance improvement we are seeing for ranges that match ~40% of the index is a

30x speedup for instance!

Given that we are intersecting a term query that matches 0.1% of the index with a range query, the expectation was that points would perform better than doc values when the range matches less than 0.1% of the index and doc values would perform better than points when the range matches more than 0.1% of the index. And hopefully the new range query that makes a decision based on the selectivity of clauses would do the right choice all the time. For this particular query, expectations are perfectly met!



Query latency for 1% term and range

This time we are running a term query that matches 1% of the index instead of 0.1% and the results are not as good: in practice using points rather than doc values for the range would have been faster up to a frequency of about 8% while we switched to doc values at 1%. It is quite disappointing that it made the wrong choice for frequencies between 1% and ~8% and greater than ~80%, but overall it still looks better to me than using either points or doc values all the time (you may disagree!). This shows why query planning is a complex and challenging problem!

Conclusion

As we saw with the benchmarks, this change has the potential of bringing serious performance improvements when ranges that match lots of documents are intersected with selective queries. We used the simple example of a conjunction that only contains two required clauses, but this change works for arbitrary deep and broad query trees: it partitions leaf queries into queries that are used to **lead the iteration** and queries that are only used to **verify matches**. Simple queries like term queries execute similarly in both cases, however ranges will now execute using points when they need to lead and doc values when they are only used to verify matches, while they used points all the time before.

Another simplification that was made in this blog post is that we focused on range queries. However this enhancement is implemented for geo bounding-box and geo-distance queries too, so you can expect improvements to query latency if you are intersecting geo queries with selective queries too!

We hope to be able to bring similar improvements to other queries that always evaluate against the entire index like `terms`, `prefix`, `wildcard` or `regexp` queries. Those are more complicated because there is no obvious way to estimate how many documents they will match up-front without paying the cost of running these queries. But hopefully we will find a way!

This change will be available in Lucene 6.5 and Elasticsearch 5.4, you can learn more about the low-level bits on LUCENE-7055 and by watching this Elastic{ON} talk. Happy searching!