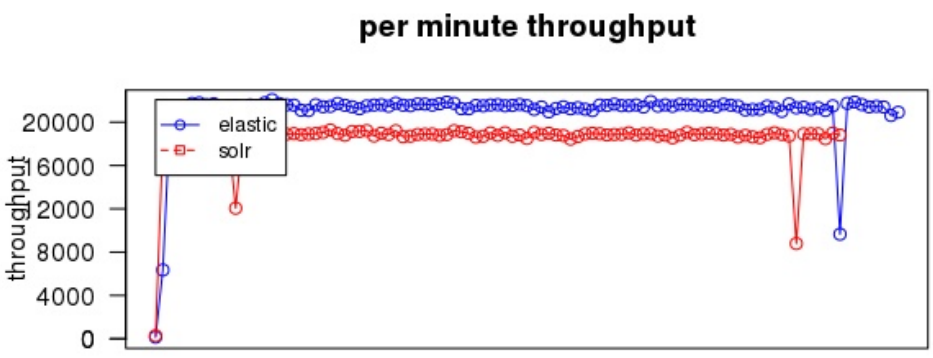
ElasticSearch vs Solr

by Glenn Engstrand

Are all Lucene based search engines the same?

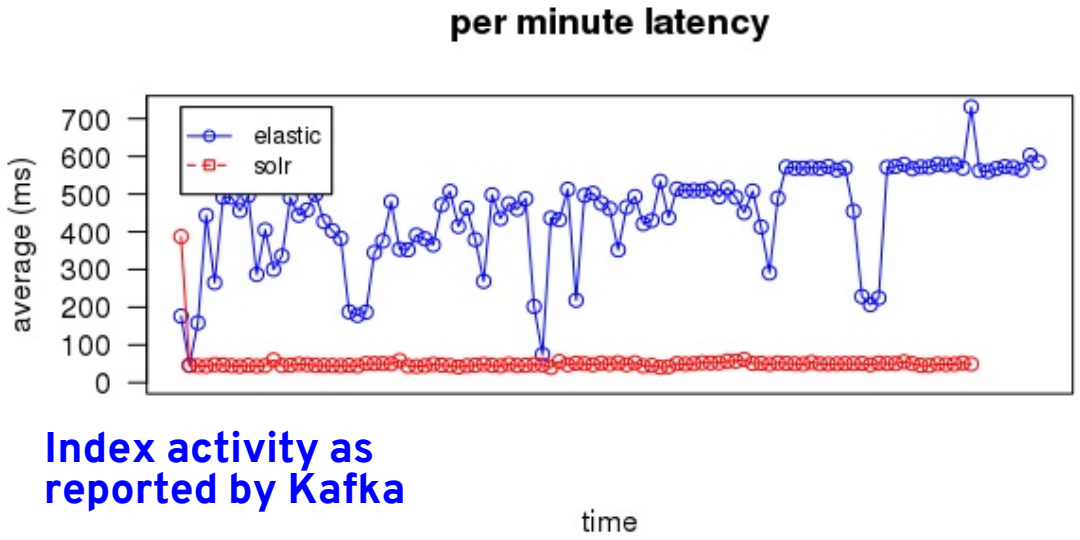
In the world of open source search projects, there are two popular web services and they are both based on Lucene; ElasticSearch and Solr. There are already plenty of online resources devoted to comparing these two technologies in terms of features and functionality. What I wanted to find out was how they compared in terms of performance under load. Nothing fancy, mind you. Just your basic, ordinary term search on documents of 150 words in length. This blog documents what I did to research that topic and what I discovered in the results.

I didn't want to make a benchmark specifically for testing ElasticSearch and Solr because the actual implementation details of the benchmark itself may subtly favor one technology over the other. Instead, I took already developed assets of a micro-service, and associated load test, that I wrote for a rudimentary news feed. That micro-service already used Solr. All I had to do was include an implementation for ElasticSearch then run two sets of load tests, one for Solr and the other for ElasticSearch. Though the micro-services involve other technologies, the only difference between the two sets of tests is choice of search technology.



Index activity as reported by Kafka

For both Solr and ElasticSearch, I used the default configuration which basically means a developer friendly single node (no clustering).



Index activity as reported by Kafka

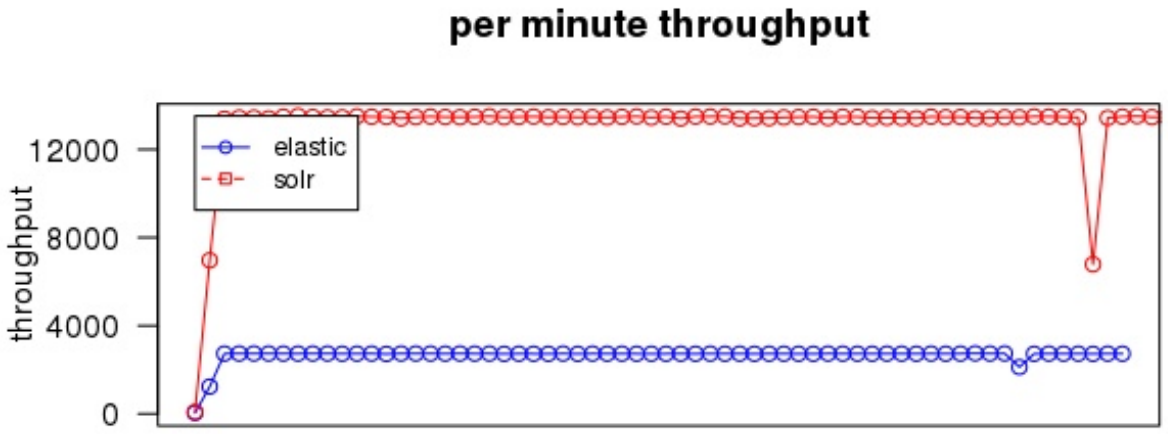
How index schemas are defined in these two technologies are different but the differences are trivial. For Solr, the schema is defined in an XML file that is loaded at time of service start. For ElasticSearch, the schema is defined with a restful PUT command. For the purposes of these experiments, how the documents were analyzed while the index was being built are very similar.

Perhaps a more significant difference may be in how the application connects to, sends documents to, and retrieves search results from the search appliance under test. For Solr, I used the SolrJ library which did not batch commit but did use soft commits. The ElasticSearch application also has a Java client library but I was not able to use it due to a version incompatibility in one of the dependent libraries. I ended up using the Apache HTTP client because it is already needed by another component so no change to the dependencies of the project. ElasticSearch does soft commits (they call it Near Real Time or NRT) automatically and both Solr and ElasticSearch are configured with a default time interval of 1 second.

Let's cover the actual experiment itself which was conducted on AWS on two different days. One day was for ElasticSearch and the other day was for Solr. On each day, I used the standard load test application which spins up three threads. The first test runs for two hours. Each thread creates lots of participants, creates friend

relationships between these participants, then posts new activities to these participants outbound feeds. It is this last part that inserts documents into the search appliance. The second test performs the search operations for an hour.

the durations for the entire request and not just the time interacting with the search appliance. Because everything else is identical, the difference in durations between the two sets of tests can be attributable to the difference in search engines.



Search activity as reported by Kafka

How is the performance for all this load testing measured? There are two approaches. The micro-service itself logs the performance relevant information for each request to Kafka. I captured those logs to a file then I filtered out all but the outbound post and outbound search operations that I analyzed with some R scripts afterwards. I also wrote an application that polls each search appliance's statistics plugin once per minute and writes the delta results to a CSV file. The Kafka data has

What were the results of the experiments? I analyzed the data for both throughput and latency and cross correlated those results between the two different sources of the data. For throughput, I found a lot of correlation between the Kafka data and each search appliance's statistics plugin. Two sources contradicted each other when it came to latency. Most probably this discrepancy is attributable to either how each statistics plugin calculates the results or in how the micro-service connected to the search appliance.

For more accurate results, strive for appliance neutrality in both your benchmarks and your measurements. Subtle differences can lead to a wide margin for error or misinterpretation.

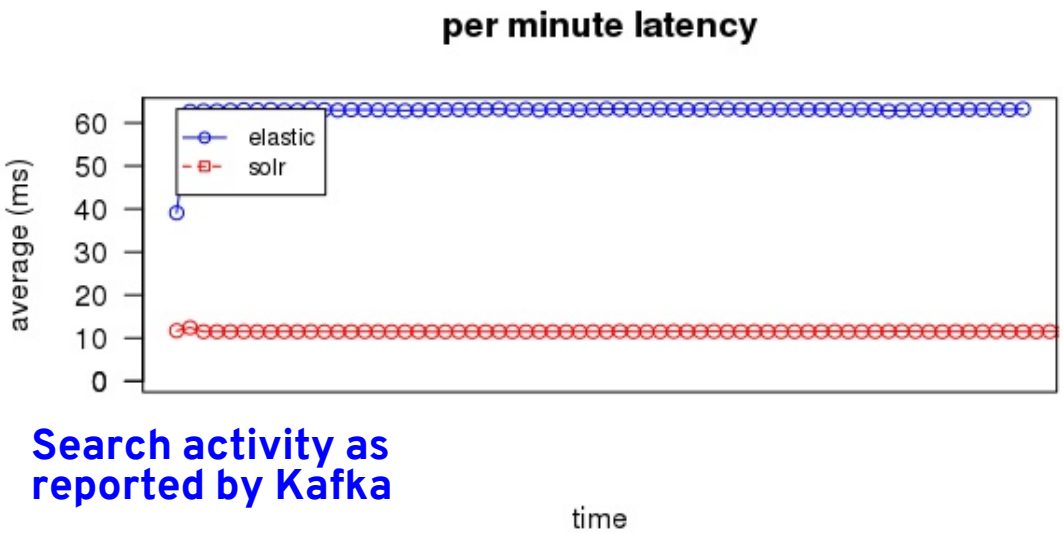
What can be safely concluded from this research effort?

When it comes to indexing new documents, Solr is the leader in latency. The Kafka data showed Solr being ten times faster than ElasticSearch. The ElasticSearch statistics plugin always reported zero latency for both indexing and searching which is hard to believe. ElasticSearch is the leader by about 13% when it comes to throughput. This is true for both approaches to measuring.

When it comes to searching documents, Solr is the clear leader in both throughput and latency. Both Kafka and the two statistics plugins recorded Solr as serving five times more search requests than ElasticSearch. According to Kafka, Solr took one fifth the time (on average) to serve those search requests than did ElasticSearch.

The ability to verify results from multiple sources of data is desirable but be wary of data that comes from the vendor whose technology is under scrutiny.

Are all Lucene based search engines the same? I think not. What I came to conclude in this investigation is that the community behind Solr has had a lot more time to refine and tune its performance characteristics in doing what the Lucene inverted index was originally created for.



Search activity as reported by Kafka