Native Text Indices in Pinot

Summary

Our text indices are currently built using Lucene. This proposal introduces a new text search engine that works on top of Pinot's native indices.

NOTE: As a part of this effort, the existing TEXT_MATCH function will not be affected, nor will its users see any changes. However, as discussed later in this document, a migration path from the current index to the native index will be supported, should the user wish to migrate.

Why Is This Feature Needed

Current text search infrastructure: Status quo, we simply build side car Lucene indices and expose a UDF which allows users to specify Lucene queries.

IMO, this is a component that should ideally be outside of Pinot since it has no correlation with Pinot itself.

So, an eventual goal is to move text search to native Pinot indices and dictionaries and follow the SQL Standard (LIKE operator) syntax as much as possible.

Now, coming to the FST itself. There are three reasons as to why a native FST makes sense:

1. Flexibility and Control -- Lucene is a full-fledged search library. It is built for generic text search use cases and consists of capabilities which allow ranked retrieval, norm storage and impact filtering, to name a few capabilities. None of these are of relevance to us since we do not perform ranking. As I mentioned before, if we are building our text search capabilities on top of Pinot data structures, then pulling in Lucene just for the FST is an overkill, and also stops us from any potential changes that we may wish to do. Lucene's FST is a generic engine, not optimized for our use cases (only dictionary IDs as output symbols, primary query load being prefix and suffix matches from LIKE operator). Other improvements may or may not come in later, but if we do not move to our native implementation, we remove the possibility of any such improvements.

2. Ability to perform Pinot specific optimizations -- As stated in the above point, it is not possible for us to do specific changes/enhancements. For e.g., it should be possible

to short circuit the evaluation of regular expressions ending with match-all and having a short prefix before the same, thus accelerating a common use case of LIKE operator.

3. Realtime Capabilities -- Lucene builds FST during segment flush, thus forcing us to flush frequently. Also, this inhibits us from doing real time text search, which is a limitation. With a native FST implementation, we should be able to explore this path.

Motivation

- Pinot's inverted indices are 5x times faster than Lucene. Having native text search capability on top of the index would be beneficial for our users
- We currently lack the ability to control our text search (i.e. Pinot committers do not get to control its direction) and it is not possible to make incremental changes or improvements to the same.
- Given that Pinot is moving to a SQL standard syntax for text search, we can make specific optimizations (for e.g., for prefix and suffix matches, not using generics) for our core text search engine
- Original plan in Pinot's roadmap was to build a native text search capability, keeping Lucene based as a stopgap

Objectives

- Build a fully functional text search engine on top of native Pinot indices, allowing exact matches, prefix and suffix matches, substring matches and regular expressions.
- Performance of the text search component (automaton, matcher and FST) should be comparable or better than Lucene's FST, matcher and automaton.
- Build the engine using core Pinot capabilities and have it deeply integrated with Pinot's core components
- Allow the library to be reusable across Pinot.
- Allow the library to be extensible without requiring application changes

Native Text Search Engine

This proposal introduces the Native Text Search Engine (Thunderbolt) within Pinot. Please note that the engine is a core part of the Pinot server code and will be maintained as a part of the same. The engine consists of the following components:

IMPORTANT: All of the functionalities supported by Lucene's FST are supported by native FST, hence all of the capabilities offered by Pinot's text engine can be supported on native FST with progressive phases.

1. Deterministic Automaton -- dynamic, run time built automaton which is used to compile and process text queries (exact matches, wildcard matches and regexp matches).

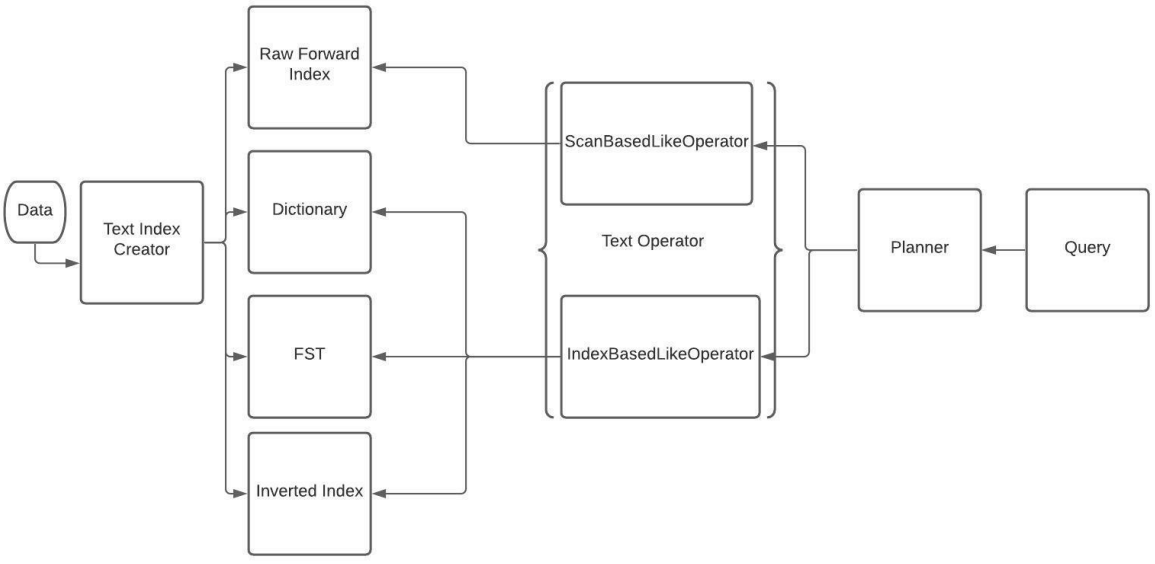
2. FST -- The FST implementation allows building transducers on top of existing dictionaries/indices. Following the current model, on each segment seal, the FST is built and stored for the segment. On index load, the FST is loaded and kept ready for searches. For details on FSTs, please see (https://www.cs.ucdavis.edu/~rogaway/classes/120/spring13/eric-transducers)

3. FST Traversal -- The library includes a FST traversal mechanism which allows exact, substring and prefix matches within the FST.

4. Regexp Matcher -- The library consists of a high performance regexp matcher which compiles a regex query to a deterministic automaton and uses the same to navigate the relevant FST to prune paths and get the result.

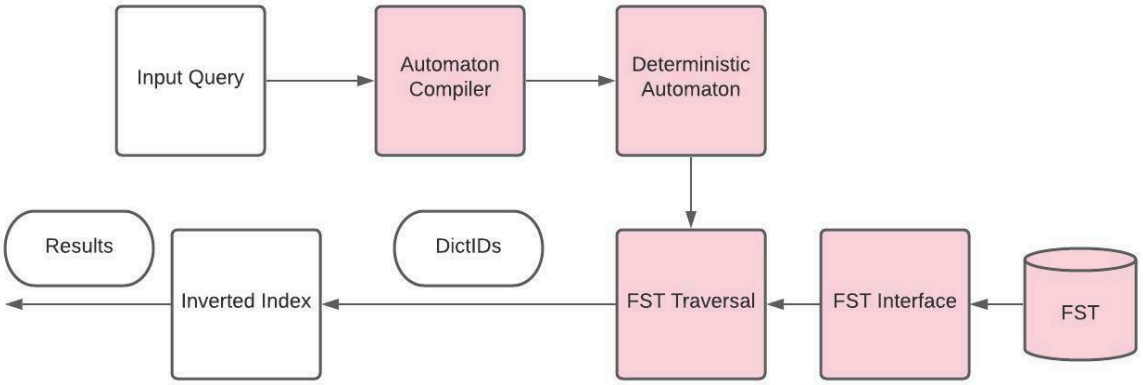
5. Builder and Serializer -- Utilities to allow building the FST and serializing it to its on-disk format.

Overall Execution Control Flow



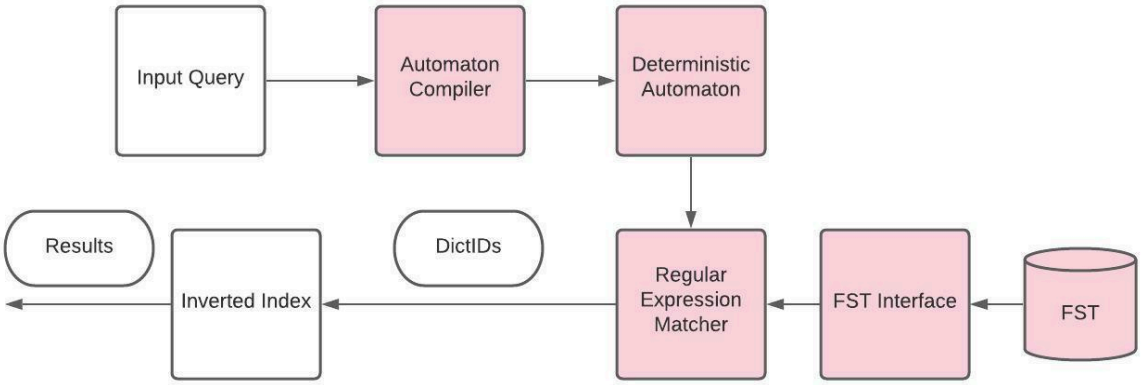
Native Text Search Operator Design

Pink boxes are Thunderbolt components



Native Regular Expression Operator Design

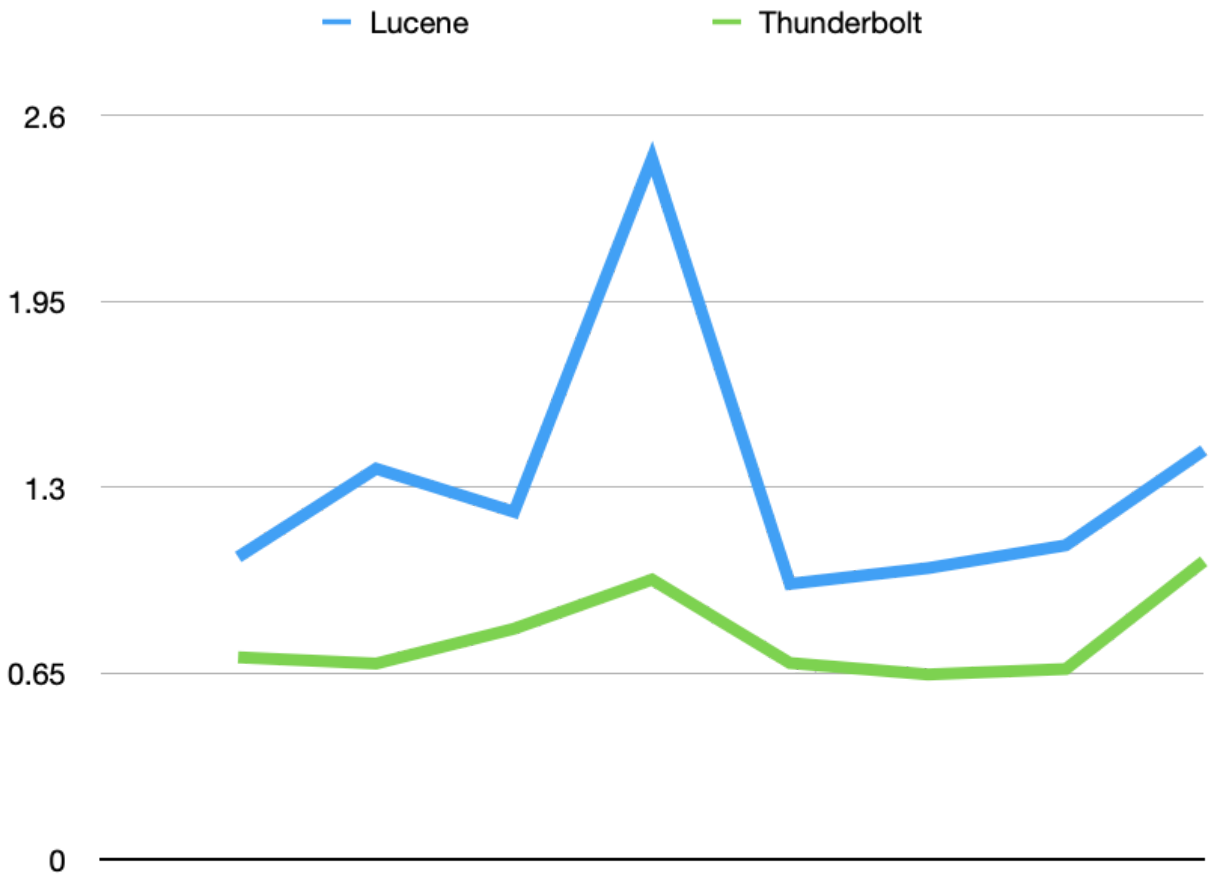
Pink boxes are Thunderbolt components



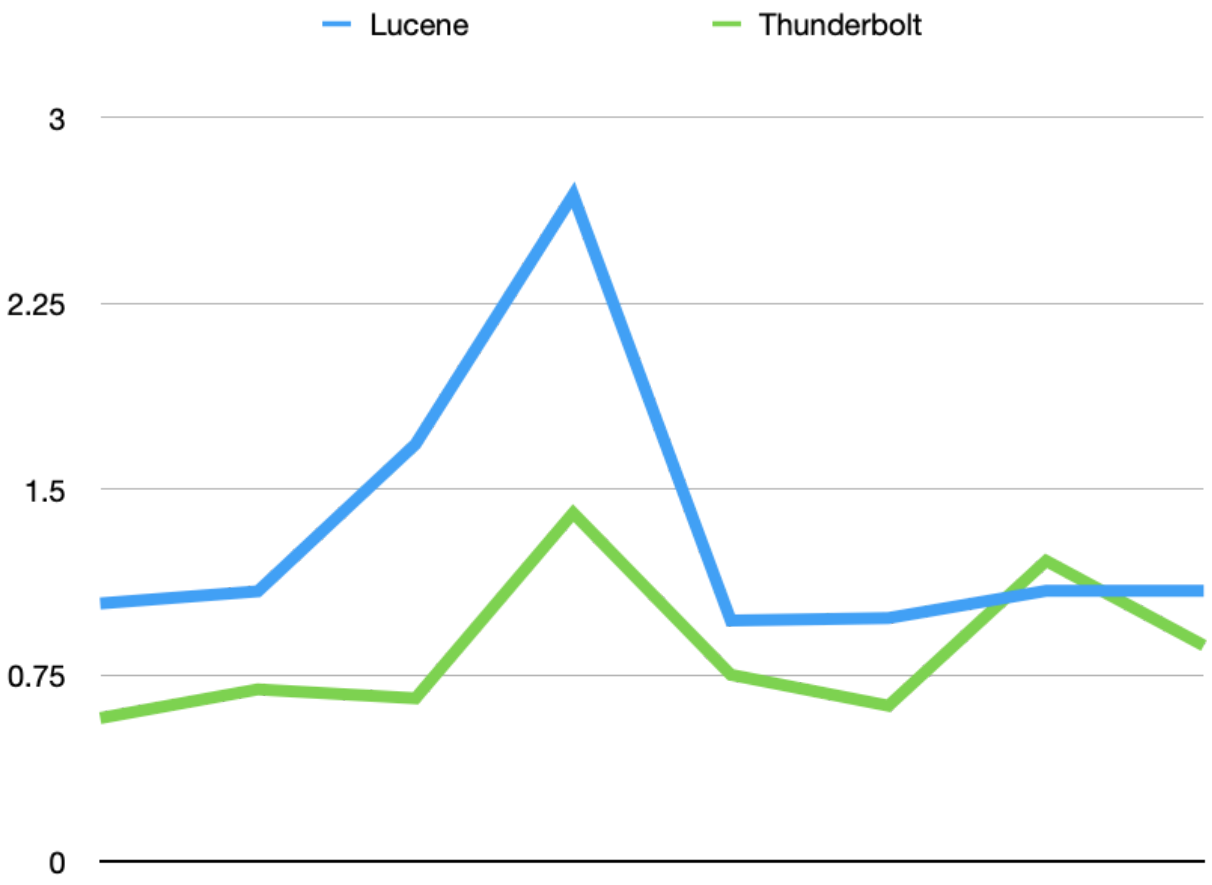
Performance Test

Test Setup -- 51 million words, with 1.5 million unique words. Each single query was run with 2 warmup iterations and 5 measurement iterations (per data point)

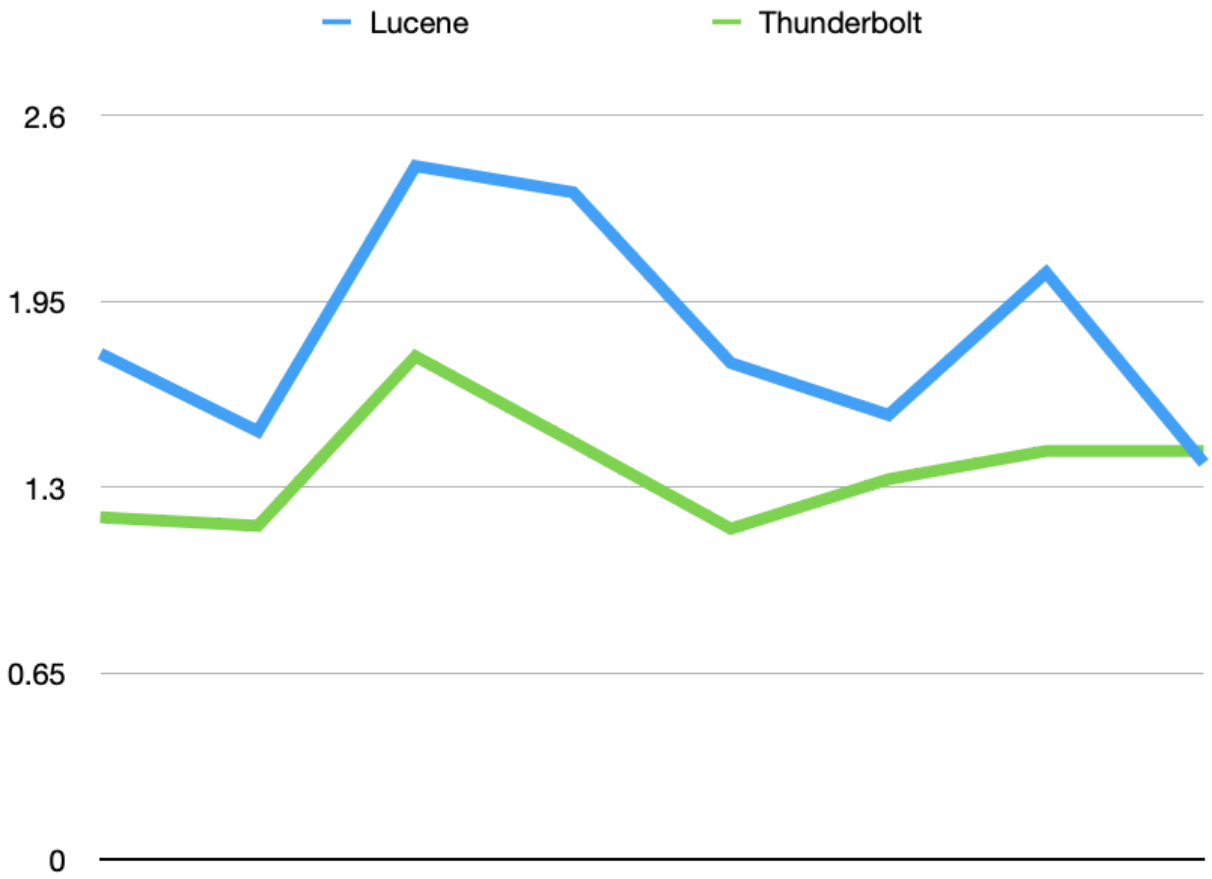
q.[aeiou]c.*



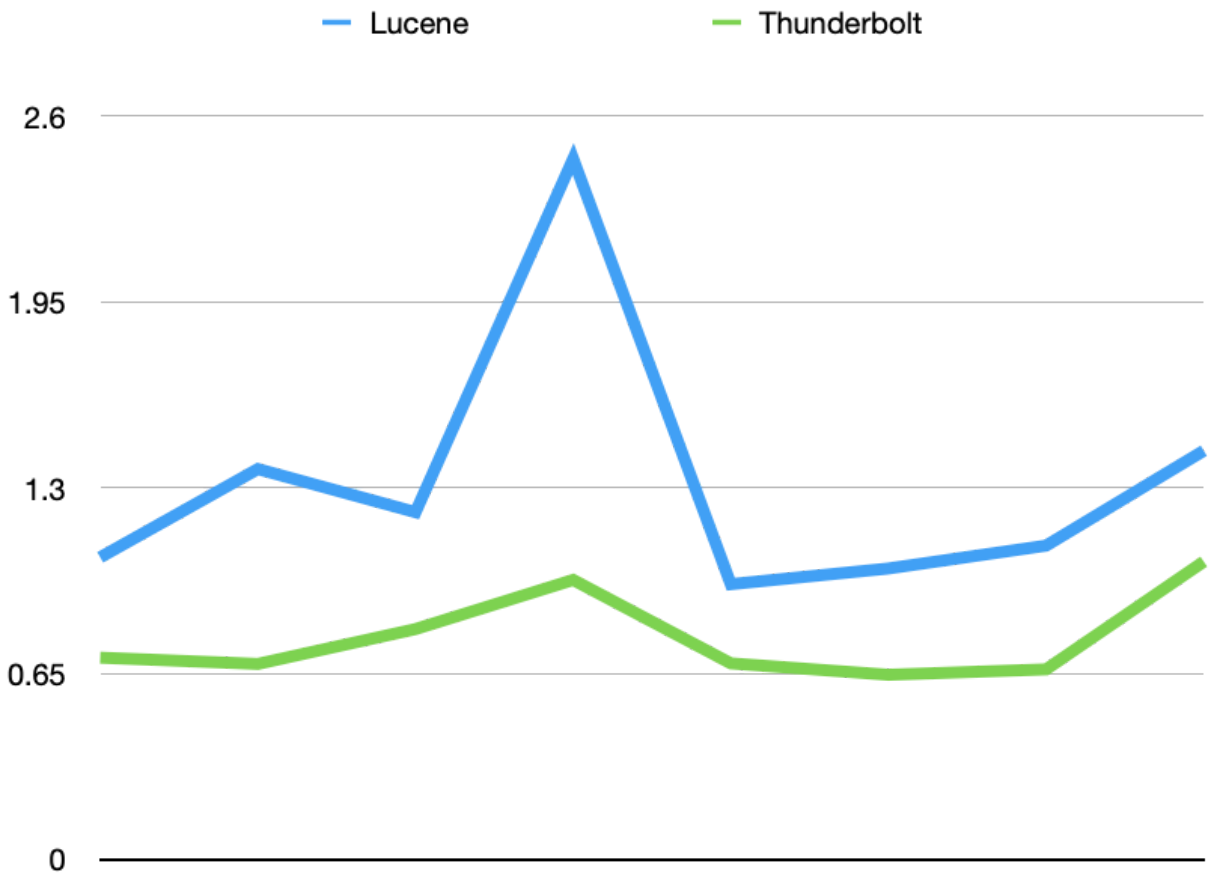
.*a



.*


Lucene
Thunderbolt
2.6
1.95
1.3
0.65
0

.*ba.*



Implementation Plan

Phase	Details
Phase 1	Core library
Phase 2	Mutable FST and related Infrastructure
Phase 3	Native index reader/writer,regex match operator on top of the library and wiring to allow new index type "native index" in table config
Phase 4	Text match operator and migrating LIKE operator to use the library
Phase 5	Fuzzy match and phrase match

Potential optimizations:

1. Offheap FST Builder -- to be considered post phase 1.
2. Prefix and suffix optimizations -- post phase 2

Migration Path

Users will be able to define a new type of index "native text index" on their columns. For existing text index fields, users will be able to migrate effortlessly since Pinot's segment processor will allow building the native text index for such fields.

Note that till completion of phase 4, we will be maintaining the existing text indices within Pinot. Native text indexes will be an independent index type that the users can choose, and Lucene based text indices will continue to operate the way they do today.

Performance Numbers of LIKE On FST Index

Average Time Test

With the support of FST indices, we collected numbers representing performance of a native based FST vs Lucene based FST.

Test Setup:

JMH Benchmark

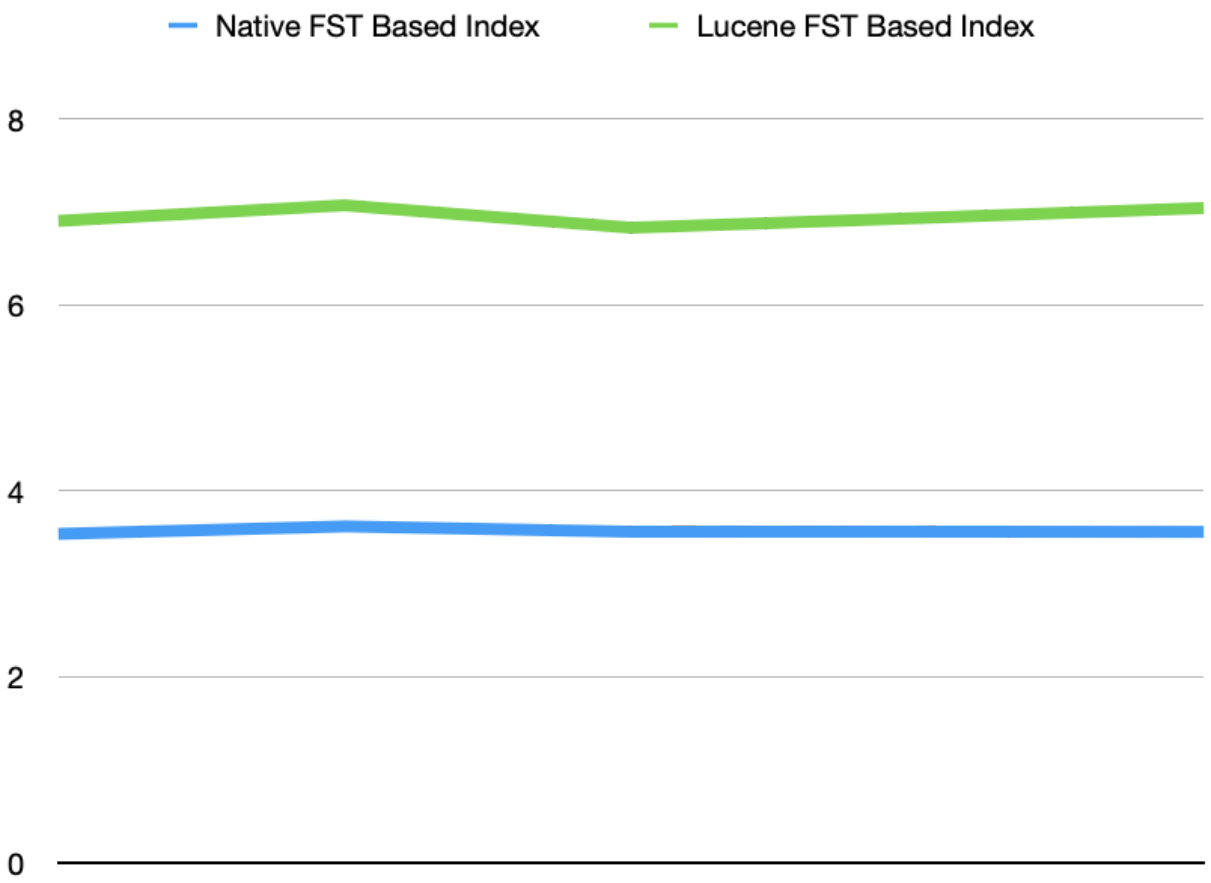
Data - 2.5 million documents, with 4 fields (2 text and 2 integers). One text field has the FST index.

Query -- "SELECT INT_COL, URL_COL FROM MyTable "
  + "WHERE DOMAIN_NAMES LIKE '%domain%'"

Hardware -- Apple M1 Pro, 32 GB Ram

Each run does a sequential execution of the above query 10,000 times. There are 3 warmup runs.

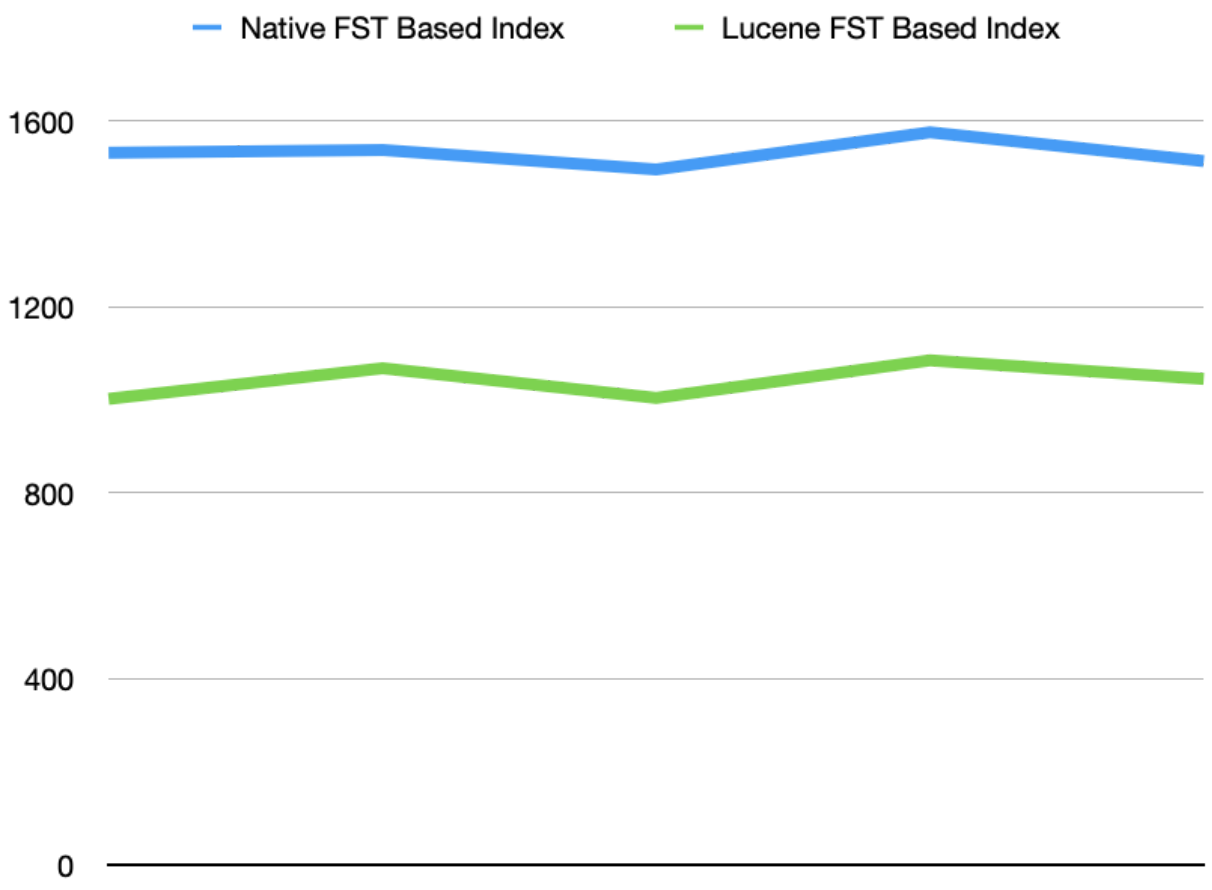
X axis represent iterations and Y axis represents latency in seconds



Throughput Test

Same test setup

X axis represents iteration and Y axis represents the throughput


Native FST Based Index
Lucene FST Based Index
1600
1200
800
400
0