Comparison of re-Isearch to Other Engines

Edward C. Zimmermann <edz@nonmonotonic.net>

Lucene Comparison (v.01)

Lucene is a very popular engine. It is the motor behind ElasticSearch, Solr, Neo4j and many other products. To compare Lucene with re-Isearch is really to compare potatoes with fish. They have quite different histories, design considerations and goals. The following short sketch attempts, however, to outline a few points since we've been asked.

1. Design target
    - Lucene was designed to be a reasonably flexible fulltext search engine with support for fields. Its a more or less traditional unstructured text search system using an optimized inverted index.
    - re-Isearch was designed to be an object (neither text nor data centric) oriented search engine for abstract objects and structural paths (including overlays). Its been designed to try to manage wildly heterogeneous formats and extract also implicit structure. re-Isearch supports XML, SGML and other formats as-if native.
    - Lucene is packaged in a number of solutions like ElasticSearch, Solr and Neo4j. They all have applications that are nearly turn-key to make it easy to get running. Solr is bundled as the built-in search in many applications like CMSs and ECMs, Cloudera's Hadoop,...
    - Re-Isearch is not a complete solution. Its not a turn-key application. Its like Lucene merely enabling technology. Despite the command line tools supplied and the available information servers (SRU/W, ISO23950/Z.2950 etc.) its really just a library and some scaffolding.
2. Market Share
    - Lucene has a sizeable market pressence. Lucene has an infrastructure of consultants. ElasticSearch is publically traded and has a market capitalization of more than 14 billion USD.
    - Re-Isearch, despite its more than 20 year history, is the "new kid on the block".
3. Java
    - Lucene is typically pure Java. Its 100% written in Java. Its more or less Java thread safe but not completely.
    - re-Isearch is written in C++. Java is provided via a SWIG created JNI (Java Native Interface) module. Since re-Isearch is written in C++ and interfaces via SWIG, application development is not limited to Java but Python (perhaps the most popular choice and by far the best developed re-Isearch interface), Tcl, Ruby and a number of other languages.

Comparison of re-Isearch to Other Engines

- Lucene needs Java to run (although there are a few forks rewriting the algorithms and structures into other languages). Java is, in our educated opinion, a fine language for developing many kinds of applications (especially given the availability of Java developing talent) but is less than ideal for database, search and retrieval.
- re-Isearch allows for applications to use its algorithms to be written in Java but does not require the use of Java. Our favorite language for writing applications to use re-Isearch is, in fact, Python and not Java.
- Java continues to have some copyright and license issues. Java SE 8 and newer public updates will no longer be available for "Business, Commercial or Production use" without a commercial license.

4. Portability
    - Since Lucene is pure Java its portable to platforms with suitable JVMs. Packages should just run from platform to platform. No need, in theory, for specific binaries beyond the JVM.
    - re-Isearch is written in extremely portable C++. It can be targeted to most platforms (Win32, Linux, Solaris, BSD etc. are available) but demands a package for each platform. Applications, of course, written in re-Isearch's Java (or other language API) are portable to any platform.
5. Threads
    - Lucene is pure Java and with the exception of the query parser and a few other bits its thread and more or less process safe. Some of it is at the cost of S/R (search and retrieval) concurrency: searches are in memory (either in whole or segments) to make them thread robust (and avoid file system and I/O deadlocks) and so don't include changes to the index during the lifespan of the class (or segment read).
    - re-Isearch is designed to handle S/R concurrency. The indexer has been designed to be able to run nearly continuously with search always in-step with the index. Its built using the POSIX threads model. re-Isearch, however, isn't 100% 'thread-safe' in the sense that the programmer can use code indiscriminately from threads. Its been designed with reasonable process safety in mind to allow for robust development of search and retrieval applications. re-Isearch is typically used in Web server environments as a service (via protocols). re-Isearch can be via JNI embedded into application servers such as Tomcat but its not advised— and especially not on Linux 32-bit systems (due to the design of the 32-bit kernel and how it uses low memory to manage memory mapped pages which together with the size of Tomcat+Apache and the max. address space leaves very little room despite available memory in RAM and swap) highly discouraged. De-coupling search from the application server increases the resilience of the application server to traffic. Java, after all, is about modularization and client-server objects.

Do threads make sense for search? See: Should one run search in threads? In a nutshell: not really and especially not with lower cost servers built around the x86 architecture and PCIe bus. They have single data ports which limit their input and output to their serial flow through the pipeline. With fast SSDs this is particularly apparent but even with slower harddisks it is noticeable as they are limited to reads within a sector. Wincherster harddrives' relatively small disk buffers favor sector-to-sector serial reads for highest throughput. Disk access will tend to use the cache less and demand more head movements (slower, more heat, more power needed etc.). In comparing queues to concurrent threads the later take more CPU and produce response times not better than the last in a sequential queue— Search performance, after all, is driven more by memory access speed and I/O latency than by CPU speed.

6. Searching multiple indexes, JOINs etc.
    - With lucene you can only search 1 index with a query. There is no means to create virtual indexes. While Lucene/Solr/Elastic don't really have virtual targets they do support something similar: Shards. Shards allow for an index to be distributed but they demand design and are not really suitable for on-the-fly collections.
        i. Each document indexed must have a unique key.
        ii. If a search discovers duplicate document IDs it selects the first document and discards subsequent ones.
    - re-Isearch supports virtual indexes and index import. One can create on demand virtual collections of indexes and transparently search them with a single query.
        i. While each record in an index must have a unique key, keys across distributed indexes or virtual collections don't need to be unique. In fact we exploit the keys to understand that the records are talking about the same object and can use it for joins.
    - With Lucene there is no means to import and merge multiple indexes into a single index.
    - With re-Isearch can also import multiple indexes into a single index.
        i. When importing one index into another, key clashes are handled by versioning. The existing record with the clashed key gets a new revisioned key. Depending upon configuration it may or my not be market as deleted.
    - re-Isearch supports also JOINs and via the object system these joins can be to RDBMSs.
7. Permitted document size and speed
    - Lucene normally indexes only the first 10,000 words of a document. When increasing this default out-of-memory errors can occur. Despite its memory demands it still seems

to index quite slow. Its slow since it indexes on a per document basis. Eating a document, spitting out its index and then merging or optimizing into the main index.

- re-Isearch indexes each and every word of a document. It does not matter how many words a document has. The amount of memory an indexing process uses is defined by the configuration and is not related to the size of the documents being indexed, the number or frequency of there terms. The more memory (as long as the system does not start constantly swaping) one gives a process, up to the total size of all the documents indexed, the faster the indexer will run but it can also run in a tiny memory footprint. The minimum memory an index process needs is the size of the largest record (it self-adjusts for this). re-Isearch can typically index, ignoring document structure, significantly faster than one could copy the documents into a tar archive. The more and faster the I/O (memory, disk etc.) the faster the indexing process. Depending upon the document format most of the time to build a database is spent not in the term indexing but in parsing the document structure into records and parsing those records into elements.

8. Field length
    - Lucene sets (by default) the max. field length by default to 10000 terms. This is to set an upper bound for the amount of memory used for indexing a single document. Since this still can lead to OOM (Out Of Memory)— especially on 32-bit Linux platforms— one is often better off reducing it to half that value.
    - re-Isearch places no limitation of max. field length. Just as a document can contain an number of terms, a field can contain any number of terms— and one can have any number of fields as well.
9. Number of Fields
    - Lucene sets a soft maximum number of fields. They generally recommend that one don't go beyond 1000. It is just not designed for more—one can up the number but generally Lucene is designed to allows a unique set of fields per document in one index.
    - Re-Isearch sets no maximum.
10. Memory Demands
    - Lucene (including Java) needs a lot of memory to run. RAM memory consumption is more or less constant at a high level during both indexing and searching activities.
    - re-Isearch can be configured to use a specific amount of memory during index. It can also self-configure itself to run in a portion of the free RAM available on the system (determined at indexing start).
    - Lucene has a high fixed memory demand for search since segments of indexes are copied into memory.

Comparison of re-Isearch to Other Engines

- re-Isearch has a low fixed memory demand. The amount of memory needed by a search is related to the size of the resulting result. The larger the number of records found and the more hits they have, the larger the memory used. Once the result set is no longer used its disposed of.
- Since Lucene is Java it is dependent upon the Java memory management (garbage collector) to manage system memory. Java tends to "hog" memory: normally takes but returns little memory to the operating system.
- re-Isearch uses memory mapping and allocates and deallocates memory to the operating system. The model has been designed to try to run in limited resources and create a minimal impact on total system performance (other programs running).

11.Exclused terms / Stop words

- Lucene normally excludes "common" words, so-called "stop words", from the index. The general use is to have these stop words also automatically chosen on the basis of their frequency in the index. The default value of 0.4 means that words that are more common than 4 per 1000 words are automatically excluded from the index.
- re-Isearch does not demand but can allow for the use of stop words. re-Isearch supports stop words on a per-language basis (language of document) and allows for distinct lists for use during indexing (exclude from the index) and search (exclude as ordinary term for search). Common practice is to index each and every term and only use stop words, if at all, during search. The term "war", for example, might not be too significant in German ("was" in English) but means quite something else in English (conflict, name of a 1960s funk band etc.).

12.Term length

- Lucene places limits on the lengths of terms
- re-Isearch is designed to handle terms/words of any length.

13.Search Terms/Wild cards/Truncated search terms

- Lucene expands wildcards to terms before even searching. Queries are re-written into a more basic form consisting of a set of logically combined TermQuery's. The standard limit on the total number of terms in a query is 1024. For example, if the indexed documents contain the terms "car" and "cars" the query "ca*" will be expanded to "car OR cars" before the search takes place.Lucene "pre-processes" steps:
    i. A sorted term enumeration is started from the term alphabetically closest to/after the given prefix (the term characters on the left). This enumerates all terms from all existing fields in the index.

ii. For each term, Lucene checks if the term actually starts with the prefix and belongs to the given field. If so the term is added to a BooleanQuery as a TermQuery with OR logic.
iii. The process produced a constructed BooleanQuery which contains exactly as many clauses as there were terms matching the prefix.

For a WildcardQuery, the process is similar in that the term value string containing wildcard(s) is also expanded to all matching terms for the given field and OR-combined using a BooleanQuery.

- re-Isearch places no similar limits but allows for wildcards in paths, terms etc. Its really a different model. The query ""ca*" will search for all the terms that start with "ca". If there is a limit defined in the search for time (which may be set to a limit or allowed to be unlimited) or number of records (which can be set to an absolute number, a number as a proportion of the total number of records in the index or unlimited) the search will stop when that mark is passed (default is unlimited number of records).
    i. re-Isearch does not expand the query into a boolean OR'd expression but searches directly for the query. This is more direct and allows for query structures to be re-used. With Lucene for each change in an index the Query must be re-parsed (Note: the Lucene query parser is NOT thread safe).
    ii. re-Isearch supports not just wildcards but full glob (POSIX 1003.2, 3.13) including some extensions. These can be applied to the entire search expression including path and term components.

- Lucene normally supports only wildcards to the right (prefix queries).
- re-Isearch supports both right and left truncation as well as generic glob expressions.

- Lucene does not normally allow for wildcards in field names
- re-Isearch allows for wildcards (glob expressions) in field names and paths.

- Lucene does not support combinations of phrase and wildcard as in the right truncated phrase search "search optim"*
- re-Isearch allows for wildcards.

14.Proximity

- Lucene does not really support proximity but a concept of "phrase query slop": the maximum number of full word "moves" required to get all the words next to each other in the proper order. For simple paired expressions like "dog cat"~10 (the ~10 specifies the moves) it comes out as within 10 (or whatever positive integer specified) words.
- re-Isearch has a concept of proximity. Distance, however, is not measured in words but bytes as in the original indexed document. This makes sense since in marked-up

documents what's a word? re-Isearch also has heuristic concepts of near and can also restrict proximity to within a common field instance.

15. Normalization

- Lucene supports both TD-IDF and BM25 Normalization. BM25 is extremely popular and has done quite well in search comparison testing.
- Re-Isearch does not support BM25. This was a decision—that may be revised—based upon the belief that it is not wholly suitable to the underlying search paradigm. BM25 adjusts the cosine metric with a guesstimate parameters K and a weight b. The weight b applies to a kind of linear weighting of the function according to a result's length with respect to the average result length in a result set. It does not consider the proximity of hits within the record. It only factors the frequency, record length, size of the result set and average size of records therein. Following a query we have a set that includes the hits and their location as well as a pointer to the record it is contained in. We, however, don't yet resolve particular information about the record's length. That information is contained in another data structure. We could look it up but it is an additional cost that seems, at this time, unwarranted. Our own tests using TREC data found Euclidean Normalization to more generally outperform BM11 and BM15—BM25 is just a weight factor that mixes these two algorithms.

  Another overriding design issue is the question: should the result be a segment of a record what length should be used?. The whole record or that length of the segment? Recall we don't always want to just retrieve the whole record that contains the hit but perhaps the relevant elements. Since larger result segments are generally selected because they are "more relevant" any weighting against an average would degrade their rank. This results in having "more relevant" items lesser scored.

16.Boolean operations

- Lucene does not use the pure boolean information retrieval model or support boolean operators but simulates some of the basic user-side functionality for inclusion and exclusion via a modal prefix model. Lucene has two term prefix ops: "+" (for "must contain"), "-" (for "must not contain"). Terms without a prefix are "can contain". It supports via query parsers an emulation of "AND" "OR" "ANDNOT". The emulation, however, is somewhat quirky and often interprets queries differently than most (other than those familiar with the quirks) would ever expect.
- re-Isearch is overloaded with operators (probably more than most people have ever heard of).

17.Unary operators

Comparison of re-Isearch to Other Engines

- Lucene has effectively no unary operators. The closest to unary operations are term boost (weight) and "fuzzy" but they are limited to use as term modifiers.
- re-Isearch has a full-blown set of operators (again from different design considerations) and includes not just a rich set of binary but also unary operators including complement, operators to sort and manipulate sets, boost weight of the expression (according to a number of models) restrict to given fields/paths etc.

18.Query Languages/Interfaces

- Lucene does not per say have a query language since it contains only terms and modifiers (+,-, weight). These may be processed in any order. There are a number of classes to try to convert other languages into Lucene's model.
- re-Isearch uses a vector/boolean information retrieval model. Queries are driven by an object oriented automata. These may be created as program objects or parses from any of a number of query languages. At the heart of re-Isearch's model is an RPN stack based language and there are a number a number of classes to convert from other query languages into RPN.

- Lucene's boolean query class limits the number of clauses (typ. 1024). This makes sense since Lucene too limits the number of terms in a query (typ. 1024).
- re-Isearch's boolean query class places not limits on the number of clauses, terms, operators etc.

- Lucene is best when not finding records. Since there are no operators and just terms and predicates that apply to them (either weight or modality) they can without worry be easily rearranged. The query: A -B C D +E for instance can be quickly optimized by the constraints "must have" E and "can't have B.
- re-Isearch is run by a query automata. It can perform many optimizations but it can't just build upon short circuits and programs will need to run their course except in some simple cases such as all terms "ANDed".

19.Does the position of the matches in the text affect the scoring?

- In Lucene: No, the position of matches within a field does not affect ranking.
- In re-Isearch: Depends upon the score normalization model selected at search time. The CosineMetric normalization model, for instance, does use the position of the matches in text to affect the scoring. This is all search time and user selectable.

20.Field differences

- Lucene lacks diagnostics. Searching even in a field that does not exist just returns no results but without reason. Since fields are *Case Sensitive* in Lucene this is a frequent source of error.

Comparison of re-Isearch to Other Engines

- re-Isearch contains diagnostics.

- Lucene's fields are *Case Sensitive*. There is, to our knowledge, no way to switch it.
- re-Isearch *by default* makes field names and paths *case insensitive* (as the case for SGML, SQL etc). Even through XML is case sensitive (and we were among those that opposed it) we are familiar with no productive XML document types and valid instances with two (or more) siblings differing only in case of their names. While possible in XML

  ```
  <organization><name>BSn</name><NAME>Edward C.
  Zimmermann</NAME></organization>
  ```

  its poor design just as there are reasons why domain names and email addresses too are not case dependent.

21. Structure search

- Lucene is a traditional inverted index fulltext engine. Its quite good at handling a limited number of fields but is inappropriate for use with arbitary trees.
- re-Isearch is not based on "Inverted file indexes" and uses other algorithms. It has no limits on the length of terms, their frequency and and can support arbitary structures and paths, including overlap.

- The granularity of Lucene (unit of retrieval) is the record as defined at the time of indexing.
- re-Isearch allows for search-time dynamic granularity. The scale of grain (sentence, paragraph, document, chapter, book, collection, source, community, hub, inter-hub bridge, sphere, inter-sphere bridge) is defined by the result of the search and by the query.

- The product of a search in Lucene is a identification for the record. To extract elements one must load the document (parse etc.) into an object model that supports addressing elements.
- re-Isearch has no no need for a “middle layer” of content manipulation code. Instead of getting IDs, fetching documents, parsing them, and navigating the DOMs to find required elements, re-Isearch lets you simply request the elements you need and they are returned directly.