Simulation of Boolean Query Implementation Strategy in Lucene

Ran Li and Xiaojin Wu

ABSTRACT

Boolean query is an important method to achieve accurate data retrieval. The deeply study of Boolean query’s implementation strategy is conducive to flexible application of Boolean query, optimization of query expression and improvement of execution efficiency. Starting with the implementation strategy of Boolean query in Lucene framework, this paper studies the representation, execution sequence diagram, logical operation rules and sub-query merging algorithm. According to the implementation strategy, the simplification rules of complex Boolean query with depth of 2 are proposed, and their correctness is proved theoretically. Finally, the validity of these laws is verified by experimental simulation.

1. INTRODUCTION

In the application of full-text retrieval, Boolean query is widely used because of its precise and efficient features, such as Bai du, Google and CNKI. Boolean query statements often mix a variety of Boolean operations, such as AND, NOT, OR. The expressions are relatively cumbersome. For most users, it’s hard to understand and flexibly apply Boolean expressions to precisely define retrieval requests. Therefore, a thorough study of Boolean query implementation logic is of great significance for flexible application of Boolean query and optimization of retrieval function.

Boolean query is the main way to realize retrieval function in Lucene. Scholars rarely study its implementation strategy. Reference [3] studies the optimization method of document collection and scoring strategy for complex Boolean query in
Lucene, and designs a performance regression prediction mechanism. Reference [4] studies the rewriting problem of Boolean query in data integration, gives two rewriting methods, and proves the reliability of the method. The implementation strategy of SQL has been deeply studied in the Reference [5]. Although these documents aren’t a direct study of implementation strategy, they are of great referential significance. Using these methods or ideas, in Lucene framework, this paper firstly studies the implementation strategy, including representation, query parsing, execution time sequence, operation rules, etc. Then, by analyzing Boolean query, the optimization law has been found, and its validity has been proved through experimental simulation.

2. BOOLEAN QUERY

In the application of full-text retrieval, Boolean query means the construction of query expressions through Boolean operators (and/or/not) on query terms to express the characteristics that users want documents to have, as in [6]. In the framework of Lucene, either a box search or an advanced query can be transformed into a Boolean query. Boolean query can also be nested in multiple layers. A Boolean query can be used as a clause of another Boolean query. Boolean query with nested relations are called complex Boolean query.

2.1 Representation of Boolean Query

In most applications, Boolean query all uses AND, NOT, OR and parentheses to connect query keywords, such as: (x OR y) AND (u OR z), where x, y, u, z are query keywords. In Lucene, Boolean query uses a new representation, assuming that there are the following query items: a, b, c, d, e, f. Documents that need to be retrieved to meet the requirements are represented as follows:

\[ Q = (a \cdot (c \cdot d)) - (e + (f)) \]  

(1)

Then Q is a typical Boolean query, where a, b, c, d, e, f are either an atomic query or a query clause. There are three Boolean relationships for sub-query, shown as in Table I, assuming a is a term.

<table>
<thead>
<tr>
<th>Name</th>
<th>Operator</th>
<th>Eg.</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUST sub-query</td>
<td>+</td>
<td>+a</td>
<td>Doc must contain a keyword;</td>
</tr>
<tr>
<td>MUST NOT sub-query</td>
<td>-</td>
<td>-a</td>
<td>Doc must not contain a keyword;</td>
</tr>
<tr>
<td>SHOULD sub-query</td>
<td>-</td>
<td>a</td>
<td>Increase the score if doc contains a</td>
</tr>
</tbody>
</table>

TABLE I. BOOLEAN RELATIONSHIP.
2.2 Analysis of Implementation Strategy

The implementation strategy is the execution plan of Boolean query in the retrieval. In Lucene, it is the process of reading inverted tables from index library and merging inverted tables according to Boolean query statements, obtaining the document set and scoring documents. Lucene's Boolean query implementation strategy is complex and efficient.

2.2.1 SEQUENCE DIAGRAM OF EXECUTION

The indexer is a component of Lucene to realize the retrieval function. For the input Boolean query statement, it will be transformed into Query object tree. The sequence diagram of retrieval execution is shown in Figure 1.

Through the analysis of Lucene documents and source code, we can get: Weight object tree is used to calculate the weight of words; Scorer object tree is constructed to calculate the score of search words; the inverted list are merged and scored by Sum Scorer objects. In this step, the inverted list is merged to get the result document set, and the final score is calculated. The result set and the score are returned to the user.

2.2.2 ANALYSIS OF BOOLEAN QUERY

As can be seen from the sequence diagram, parsing a Boolean query and generating a query object tree is the first step in executing process. For the query represented by Formula (1), it can be interpreted as the query tree shown in Figure 2. Each sub-tree is called a sub-query.

Each leaf node of the query object tree is a query Term, also known as atomic query. The Term matches directly in the index dictionary to obtain the inverted list.
of documents containing the Term. For Query object tree, Lucene first merges the sub-trees, then merges the upper sub-trees until the root. This process is shown in Figure 3.

![Figure 2. Query Object Tree.](image1)
![Figure 3. The Sequence Diagram of Query Object Tree.](image2)

### 2.2.3 BOOLEAN OPERATION RULES

The merging of queries in Query object tree is a typical logical operation. The rules for merging sub-queries of the same level in the query tree are shown in Table II.

The merging operation of sub-queries is essentially the merging operation of inverted list. Generally speaking, there are three main types: intersection operation, merging operation and subtraction operation. For three operations of set, Lucene uses multi-path merging algorithm, as in [7]. Multiplex merge operation is that multiple sub-queries participate in merge operation at the same time. Take intersection operation as an example:

<table>
<thead>
<tr>
<th>Sub-query</th>
<th>E.g.</th>
<th>Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>multiple MUST clauses</td>
<td>(Q = (+a +b))</td>
<td><img src="image3" alt="Diagram" /></td>
</tr>
<tr>
<td>Multiple SHOULD clauses</td>
<td>(Q = (a \ b))</td>
<td><img src="image4" alt="Diagram" /></td>
</tr>
<tr>
<td>MUST clause and SHOULD clause</td>
<td>(Q = (+a \ b))</td>
<td><img src="image5" alt="Diagram" /></td>
</tr>
<tr>
<td>MUST clause and MUST_NOT clause</td>
<td>(Q = (+a \ -b))</td>
<td><img src="image6" alt="Diagram" /></td>
</tr>
<tr>
<td>SHOULD clause and MUST_NOT clause</td>
<td>(Q = (a \ -b))</td>
<td><img src="image7" alt="Diagram" /></td>
</tr>
<tr>
<td>MUST clause, SHOULD clause and MUST_NOT clause</td>
<td>(Q = (+a \ b \ -c))</td>
<td><img src="image8" alt="Diagram" /></td>
</tr>
</tbody>
</table>
Assuming that \( Q = ( +x +y +z ) \), \( x, y, z \) are sub-queries, Lucene uses a multi-path fast merging algorithm based on jump table. The inverted chain table of initial state is shown in the first part of Figure 4. First, the list of \( x, y, z \) is sorted by length, starting with the first element of the shortest list \( y \), and then searching in the list \( z \), as shown in the second part of Figure 4. When the same element is found in \( z \), then the same element is found in \( x \), then this element is hit, as shown in the third part of Figure 4; so on, until the end of the list \( y \).

![Figure 4. Multi-path Fast Merging Operation.](image)

In the mixed Boolean operations of three clauses, MUST clauses are combined firstly, then MUST_NOT clauses, SHOULD clauses finally. Therefore, for a query \( Q \), there are multiple sub-queries, which are expressed as follows:

\[
Q = (+a -b c +d +e), a, b, c, d, e \text{ are sub-queries;}
\]

According to the priority rules and the operation rules in Table II, the merging operation of \( Q \) is as follows:

\[
Q = ( ( +a +d +e ) -b ) \ c);
\]
\[
Q = ( +Q1 -b ) \ c), \ Q1 = (+a +d +e);
\]
\[
Q = (+Q2 \ c), \ Q2 = (+Q1 -b);
\]

3. ANALYSIS OF IMPLEMENTATION STRATEGY

Implementation strategy of Boolean query directly determines the performance of Lucene retrieval, and further study of the implementation strategy is conducive to finding ways to optimize the retrieval performance. This paper puts forward possible optimization strategies.

Assume a Boolean query \( Q1 = (+a+b+(+c+d -e) - f) \), where \( a, b, c, d, e, f \) are sub-queries, and the query tree is shown in Figure 5. This is a query tree with a
depth of 2. Firstly, the sub-trees are merged, and then the upper sub-trees are merged until the root node. According to the execution logic, the query tree in keywords needs to perform four times of multi-path merging operations and produces three intermediate results. If $Q_1$ is simplified to the $Q_2$ query tree shown in Figure 6, only two times of multi-path merging operations are needed and one intermediate result is produced. In theory, query efficiency and spatial comp lexity will be greatly improved.

![Figure 5. Q1 Query Tree.](image1)

![Figure 6. Simplified Query Tree.](image2)

Question: are Boolean queries represented by $Q_1$ and $Q_2$ equivalent? The answer is yes, as evidenced by the following:

Let $A, B, C, D, E, F$ correspond to the result set of sub-queries $a, b, c, d, e$ and $f$ respectively, then:

$Q_1 = A \cap B \cap (C \cap D-E)-F$

$= A \cap B \cap (C \cap D \cap \overline{E})-F$

$= A \cap B \cap C \cap D \cap \overline{E} -F$

$= A \cap B \cap C \cap D-E-F$

$Q_2 = A \cap B \cap C \cap D-E-F;$

Therefore, $Q_1 = Q_2$, the certificate is completed.

The author studied the interfaces of full-text retrieval systems such as CNKI, Baidu and Google, and found that when the user's retrieval requirement were resolved into query tree, the depth is 2. Therefore, the author deeply studies the Boolean expression of user's query requirement, if the expression or part of the expression is simplified, the number of query merging operation and the intermediate results will be reduced. In this paper, Boolean expressions with depth of 2 are studied in depth according to set operation rules. Table III lists the law with regularity and simplicity.
TABLE III. SIMPLIFICATION LAW.

<table>
<thead>
<tr>
<th>Clause</th>
<th>The Law</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUST</td>
<td>* If the clause contains MUST atom query and doesn’t contain SHOULD atom query, it can be simplified equivalently.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>( Q= (+a + (+c +d - e)) )</td>
<td>( = (+a + c +d - e) )</td>
</tr>
<tr>
<td></td>
<td>* If the clause contains MUST and SHOULD atom query, it can be simplified if the ranking problem is not considered.</td>
<td>( Q= (+a + (+c +d +e)) )</td>
</tr>
<tr>
<td></td>
<td>* If the clause contains only one SHOULD and several MUST_NOT atom queries, they can be simplified equivalently, but SHOULD become MUST.</td>
<td>( Q= (+a + (d - e - f)) )</td>
</tr>
<tr>
<td>MUST_NOT</td>
<td>* If the clause contains two or more MUST atomic queries, it cannot be simplified.</td>
<td>( Q= (+a - (+c +d)) )</td>
</tr>
<tr>
<td></td>
<td>* If there are only one MUST and several SHOULD sub-queries in the clause, they can be simplified equivalently. MUST become MUST_NOT, SHOULD be dropped.</td>
<td>( Q= (+a - (+d +e +f)) )</td>
</tr>
<tr>
<td></td>
<td>* If there are only a few SHOULD atom queries in the clause, they can be simplified, and SHOULD become MUST_NOT.</td>
<td>( Q= (+a - (d +e +f)) )</td>
</tr>
<tr>
<td>SHOULD</td>
<td>*There is no law</td>
<td></td>
</tr>
</tbody>
</table>

4. EXPERIMENTAL SIMULATION

4.1 Simulation Design

In order to further analyze the merging operation rules and execution efficiency and verify the optimization strategy, this paper constructs an experimental environment, simulates various optimization schemes, and compares the query results and execution efficiency. The Lucene version is 4.10.3, and the word analyzer is IKAnalyr02012_u6. The index database is constructed by 140,000 commodity order records. Each record contains the number, commodity name, type, address, consignee and price fields, which are mapped to the number, name, type, address, user and price fields in the Document object respectively. All experimental data are recorded under the condition of stable computer load. Five key words were selected in the experiment:

\( A = \) name: "male",
\( B = \) type: "venue",
\( C = \) name: "cotton",
\( D = \) type: "underwear",
\( E = \) name: "genuine".

All the query experiments are based on the combination of the above five keywords.
4.2 Simulation Results

According to the two types and five optimization rules listed in Table III, Boolean expressions are constructed to retrieve the results. The results, time consumption and ranking records are shown in Table IV.

<table>
<thead>
<tr>
<th>Clause</th>
<th>Query Expression</th>
<th>Result Count</th>
<th>Time (ms)</th>
<th>Simplified Expression</th>
<th>Result Count</th>
<th>Time (ms)</th>
<th>Optimized proportion</th>
<th>Sorting Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>MUST</td>
<td>(+a+b+(c+d–e))</td>
<td>554</td>
<td>233</td>
<td>(+a+b+c+d–e)</td>
<td>554</td>
<td>208</td>
<td>10.7%</td>
<td>no change</td>
</tr>
<tr>
<td></td>
<td>(+a+b+(c d e))</td>
<td>1423</td>
<td>337</td>
<td>(+a+b+c d e)</td>
<td>1423</td>
<td>324</td>
<td>3.8%</td>
<td>some change</td>
</tr>
<tr>
<td></td>
<td>(+a+b+( c–d ))</td>
<td>869</td>
<td>241</td>
<td>(+a+b+c–d)</td>
<td>869</td>
<td>237</td>
<td>1.6%</td>
<td>no change</td>
</tr>
<tr>
<td>MUST_NOT</td>
<td>(+a+b -( +c d ))</td>
<td>9716</td>
<td>846</td>
<td>(+a+b -c)</td>
<td>9716</td>
<td>783</td>
<td>7.4%</td>
<td>no change</td>
</tr>
<tr>
<td></td>
<td>(+a+b-( c d ))</td>
<td>8422</td>
<td>724</td>
<td>(+a+b -c-d)</td>
<td>8422</td>
<td>751</td>
<td>3.6%</td>
<td>no change</td>
</tr>
</tbody>
</table>

From the experimental data in Table IV, it can be seen that the Boolean expression optimization rule listed in Table III is correct, and the execution efficiency of optimization is improved. Because of the small amount of experimental data, the document which come from order data is very small, the query expression is relatively simple, the machine load is relatively low, so the improvement of execution efficiency is not obvious. With the increase of data volume and query load, the efficiency of optimized query execution will inevitably be greatly improved.

5. CONCLUSIONS

This paper deeply studies the implementation strategy of Boolean query in Lucene framework. According to the logical process of Boolean query, the simplification rule of complex Boolean expressions is analyzed, and a type in simplified rules has been proved. Through theoretical analysis and simulation, the simplification rule of complex Boolean expressions is correct. Moreover, the simplification of complex Boolean operations effectively improves the retrieval efficiency.
ACKNOWLEDGEMENTS

Funding project: Research on Quality Evaluation of Enterprise Knowledge Bae (B2018237), Scientific Research Project of Hubei Education Department.

REFERENCES