International Journal of Advanced Computer Research (ISSN (print): 2249-7277 ISSN (online): 2277-7970)
Volume-4 Number-1 Issue-14 March-2014

Performance evaluation of searching using various indexing techniques in Lucene with Relational Databases

Chetan Khilosiya[1], H. P. Channe[2]

Abstract

The Organizations commonly use relational databases for transaction processing, but big portion of database operations involve select operation. As data grows beyond few million records selection tends to take much time in whole transaction. One approach is to build indexes in database on columns which are frequently used in selection. If there are more than one table (which is general case) selection takes more time. Another approach is to use searching framework for searching records. Apache lucene is very popular, fast open source searching framework used in many projects. So here we are trying to evaluate use of lucene searching to find records fast so as to get performance benefits from lucene's fast searching capabilities and offload selection work from databases. We will evaluate different indexing types in lucene to see which best fits to our need. At last we evaluate that is this arrangement can provide performance benefits, and which index type is best suited for that.

Keywords

Performance evaluation, Search Process, Indexing methods.

1. Introduction

Relational databases are required and optimized to support ACID properties. Selection operations are very common, but don't require ACID property. There are many organizations that do less insertion/update queries related to selection queries. For large databases selection tend to take much time and resources. To get more performance more servers can be added, but cost off licensing increases as servers increase for popular commercial databases. On the other hand searching frameworks are designed to provide better performance for retrieving records, on expense of requiring more time for insertion and updates. As organization need relational database to provide transaction management support they can't replace databases with searching framework which provide better performance for searching. So by using external searching framework we can offload work of fetching records from databases, so that databases remain free to do other transactional work. We are using Apache Lucene framework to provide searching facility [1]. We are trying to evaluate the framework in which the records are searched in lucene [2], and then they are used in database operations. For our analysis we are using MySQL database.

Lucene internally stores indexing in form of documents. The documents internally contain fields. Each field has field name. With each field name its value is stored. In a document we can add multiple fields. So for searching we can search on a field of documents. So while adding database entries from mysql database, we treat each row as a document and each column as fields. Each document is stored with its unique document ID. While searching lucene only provides document IDs. Using document IDs we can retrieve documents. The document IDs are unique but not permanent. We can add documents and delete documents entries also. So the IDs for deleted documents do remain unused. Periodically lucene performs compression of index in which the document IDs of deleted documents are collected for reuse, at this time the document IDs change, that's why one cannot rely on lucene document Ids [3]. There is another framework apache solr which is a open source enterprise search platform from the apache lucene project but it is used for web searching purpose. Solr internally uses lucene for indexing and searching purpose and builds the server above that to provide scaling the searching using index replication on multiple servers. The centre server adds the indexing entries and then the server replicates the indexes. The searching can be performed using any replication servers.

Manuscript received March 07, 2014.
Chetan Khilosiya, Computer Engineering, University of Pune, Pune, India.
H. P. Channe, Computer Engineering, University of Pune, Pune, India.

2. Related Work

Lucene provides many indexing types. We are using SimpleFSDirectory which uses JAVA IO API to store index on hard disk, NIOFSDirectory uses JAVA NIO API to store index on hard disk, and RAMDirectory which uses physical memory to store index. RAMDirectory gives best performance for small indexes; it has limitation of physical memory available on system [1]. Lucene shows search results in relevance order, so more relevant results shown first. The automatic indexing is performed on documents, and the vector of document containing the words in documents and the weight of the word in document [4]. We can calculate the precision and recall of the term search. But in our requirement relevance is not required as all records are necessary for transaction processing. Lucene search queries can be used to search on multiple fields. The automatic text classification can be done using pre-classified documents set and using machine learning. But this method produces vectors which are not human understandable. The author gives genetic algorithms to classify documents and produce classification which is human readable [5]. Even in lucene we can provide fuzzy search [6] in which small error in input can be tolerated [7]. Lucene divides each input string in tokens (single word). Multiple word input is divided into tokens to search. Multiple keywords can be used on same column to search. Complex identifier indexing is now in active research area [8], but it is not supported by lucene.

In 2011, Guoliang Li et al. [9], studied different approaches for type-ahead search. As users enters query, every keystroke generates a new query. They are using fuzzy search to mitigate minor errors. They use tree with inverted lists at leaf nodes as data structures. In 2011, Jimmy Lin et al. [10], proposes to use full-text indexing for map-reduce framework to optimize selection operations on text fields within records. Results show moderate improvement in query processing time and processing time savings at worker nodes. In 2003, James Abello et al. [11] shows indexing mechanisms can also be used for graph databases. They propose hierarchical two-level indexing schema called gkd*-tree, which composed of first-level kd-tree index with second-level of redundant R*-tree that indexes leaf pages of gkd-tree. In 2001, Maayan Geffet et al. [12] create Bibliography on Web project and uses hierarchical index to which entries are linked. So search results would give hierarchy of results of relevant topics. In 2005 H. V. Jagadish et al. [13] presents a efficient B+-tree based indexing method for k-nearest neighbour search in high-dimensional metric space. Data partitions are flattened into single dimensional value for indexing and KNN-search performed using range search. In 2005, Paolo Ferragina et al. [14] propose two compressed data structures for full-text indexing so that while searching, decompression of data would not require. So that data storage will be less and overall processing required for searching will be less. In 2012, Rushdi Shams et al. [15] propose that using text denoising method that extracts denoised text, the indexer performs better than full-text trained indexer. Text denoising can reduce text size up to 30% of original size. Nutch is a search engine which is very scalable and uses apache lucene as core indexing technique. In 2007, Jose E. Moreira et al. [16] analyse performance and scalability of various configurations of nutch.

3. Programmer's Design

We created three different indexes, and mysql database to evaluate record fetching performance. Below is workflow diagram.

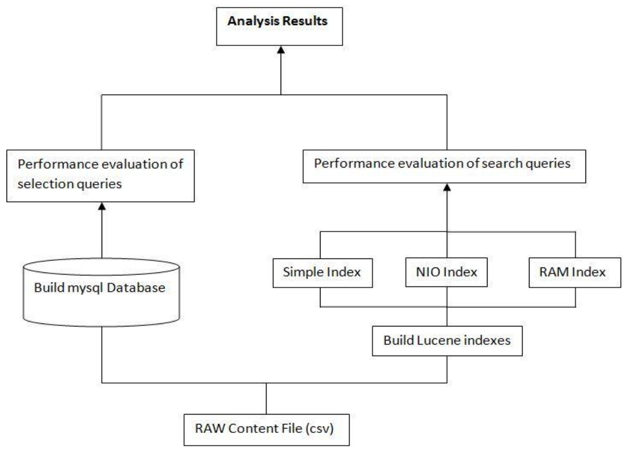

Figure 1: Workflow of Evaluation

Mathematical Model

S = System

S = {I, O, BD, BLI, PED, PEL, A}

Input:

I = {RF}

RF = Input raw file.

Output:

O = {DB, SI, NI, RI, R}

Where,

DB = mysql Database.

SI = Simple File System Directory Index
NI = NIO File System Directory Index
RI = RAM Directory Index
R = Result of Analysis

Functions:

- BD (RF) ∈ RF → DB
  Build mysql database from raw file data.
- BL (RF) ∈ RF → SI, NI, RI
  Build indices in lucene from raw file data.
- PED (DB, Q) ∈ DB, Q → Td
  Get query time (performance measure) for selection queries from mysql database.
- PEL (SI, NI, RI, Q) ∈ SI, NI, RI, Q → Tl
  Get performance measure in lucene for different indexes.
- A (Td, Tl) ∈ Td, Tl → R
  Get Analysis Result from two performance measures.

Below given the system configuration used to take performance measure.
Processor: 1.7 GHz, Core i5
RAM: 4GB, Hard disk: 160 GB,
OS: Fedora 17, java environment: openjdk-1.7.
Apache Lucene 3.6, mysql 5.5
The last RAM index stored is the method in which we store index on hard disk as well as in RAM. So next time when we restart server the index can be loaded in RAM which is very fast compared to build index again. The database system used is mysql.
We use default 16MB cache for indexes. Lucene query for user Lara smith shows 3 results than 1 shown in mysql. This is due to lucene shows extra results which contain search keyword as sub keyword. But exact match keywords are displayed due to lucene scoring formula [3].

Σ_t (t f(t in d) idf(t) boost(t field in d) lengthNorm(t field in d)) coord(q,d) queryNorm(q).

We can formulate different lucene query which only do exact match on keywords so that it will give same results as mysql and we don't need to process excess results.

Dynamic Programming and Serialization

As different users can give different, many users have part of the query common.

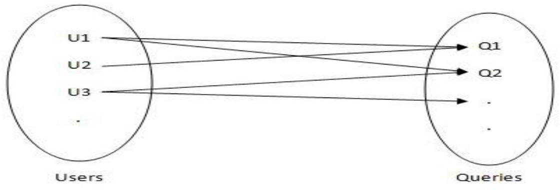

Figure 2: Mapping function of users to queries

Each query is then executed on all indexes to evaluate performance of each index.

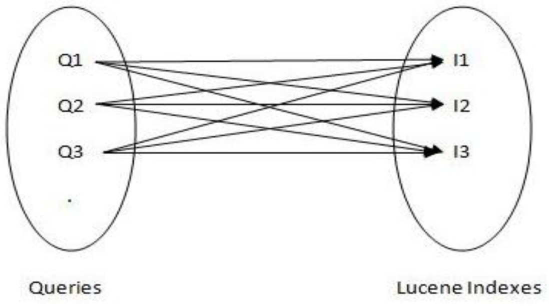

Figure 3: Mapping function of queries to Lucene Index

Data independence and Data Flow Architecture

The user uses two searching mechanisms one is mysql search and other is lucene search. Mysql search uses mysql database to store data and search. Lucene search uses lucene indexes to search records. Administrator builds database from input of raw data. And also build indexes from same data.

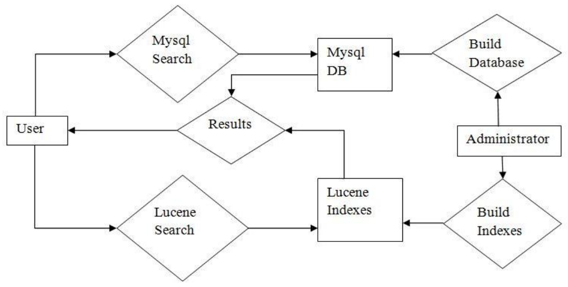

Figure 4: Data Flow Architecture

4. Results and Discussion

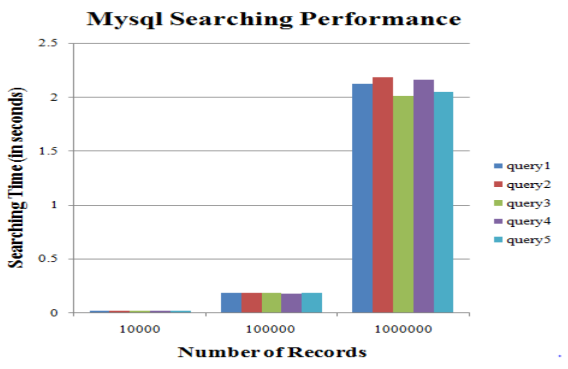

Figure 5: MySQL searching performance

We used 1 million records general user information with 28 columns, which is inserted in mysql as well as used to create index in lucene. The user ID is used as primary key to identify particular user information record. The searching is performed in three phases using 10,000 records, 100,000 records and 1 million records. Figure 5 is showing the mysql searching performance for three phases. Regardless of search query the searching time in mysql is nearly constant, because mysql searches entire table for any search query which requires the same time for same number of records. The sharp increase in searching time shows that searching time is linearly proportional to number of records. Searching time is not related to how many number of records matches our search query.

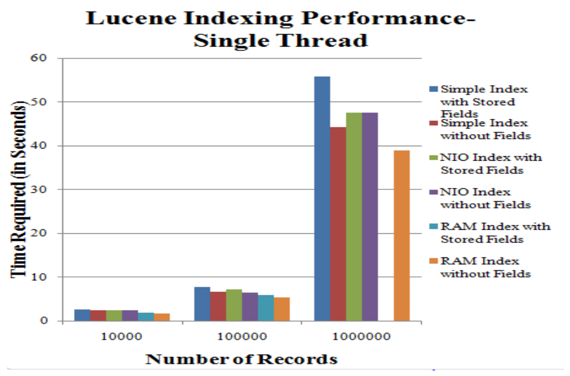

Figure 6: Lucene Indexing performance using single thread

Figure 6 shows Lucene Indexing performance while running single thread for indexing. It shows that RAM Index takes less time that Simple Index and NIO Index.

Figure 7 shows Lucene Indexing performance while running 4 threads. Building Lucene Indices required slightly less time than single threading approach. With multithreading approach size of Index increases slightly. As RAM index stores data completely in physical memory the RAM index with stored fields is not practical for indices greater than few hundred megabytes in size.

Figure 8,9,10 shows Lucene Searching Performance for 10000, 100000, 1 million records. It shows that overall lucene takes very less time for searching than mysql database. RAM Searching provides best searching time for any type of query. But it has limitation of physical memory present on the system.

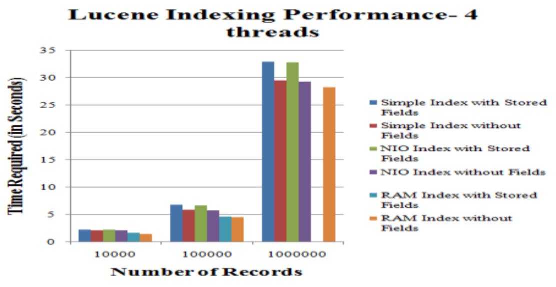

Figure 7: Lucene Indexing performance using 4 threads

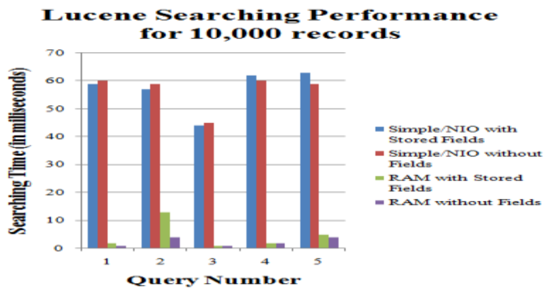

Figure 8: Lucene searching performance for 10,000 records

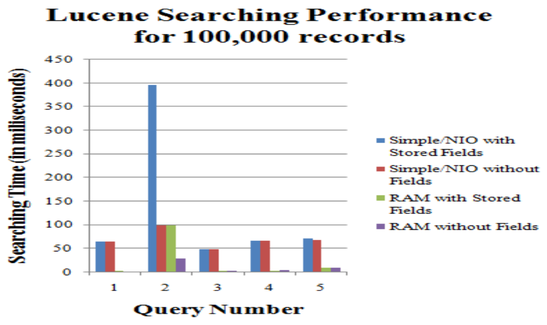

Figure 9: Lucene searching performance for 100,000 records

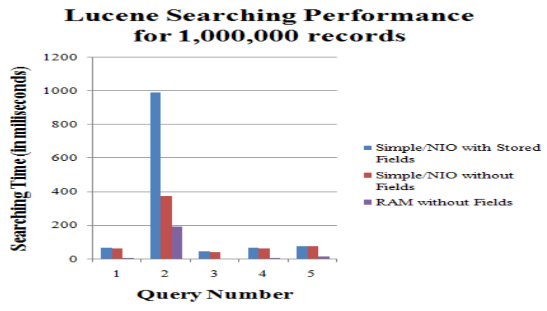

Figure 10: Lucene searching performance for 1 million records

Lucene searching time depends on how many records present in index and how many records matches the search query. Searching time for stored field indices is slightly greater than searching time for indices without stored fields.

5. Conclusion and Future Work

Lucene searching is very fast, so we can use it to retrieve records in large databases. Although inserting is slow multiple threads can be used to build index. Index with data stored give less performance than index without data stored, and in our scenario data storage is not required as we can retrieve data from database. Multithreading approach saves some time for building lucene indices. The difference between single thread approach and multithread approach increases as the preprocessing for data increases. Multithreading approach recommended only if the data needs to be processed heavily before adding to index. Lucene index does not provide real time searching. So search results are retrieve older documents. We can evaluate the near real time searching in future to check if this can provide good solutions for retrieving recent documents in search results.

References

[1] Apache Team , “Apache documentation”, Online At http://lucenen.apache.org/core/3_6_1/index.html (as of 7 January, 2014).
[2] Deng Peng Zhou, “Delve inside the Lucene indexing mechanism”, Online At http://www.ibm.com/developerworks/library/wa-lucene/ (as of 14 January 2014).
[3] Michael McCandless, Erik Hatcher, Otis Gospodnetic, "Lucene in Action" 2nd edition, Manning Publications Co, 2010.
[4] C. T. YU, G. Salton, "Precision Weighting - An effective Automatic indexing method", journal of Association for Computing Machinery, Vol. 23, No 1, January 1976.
[5] Laurence Hirsch, Robin Hirsch, Masoud Saeedi, "Evolving Lucene Search Queries for Text Classification", GECCO'07, July 7-11, 2007, London, England, United Kingdom. Copyright 2007 ACM 978-1-59593-697-4/07/0007.
[6] Amol Sonawane, "Using Apache Lucene to search text" Online At “http://www.ibm.com/developerworks/ opensource/ library/os-apache-lucenesearch/” (as of 11 December 2013).
[7] Shengyue Ji, Guoliang Li, Chen Li, Jianhua Feng, "Efficient Interactive Fuzzy Keyword Search", Apr 20-24, 2009, ACM 978-1-60558-487-4/09/04.
[8] Gerard Salton, "Automatic text indexing using complex identifiers", 1988 ACM 0-89791-291-8.
[9] Guaoliang Li, Shenguye Ji, Chen Li, Jianhua Feng, “Efficient fuzzy full-text type-ahead search”, The VLDB Journal (2011) 20:617–640 DOI 10.1007/s00778-011-0218-x.
[10] Jimmy Lin, Dmitriy Ryaboy, Kevin Weil, “Full-text indexing for optimizing selection operations in large-scale data analytics”, ACM, San Jose, California, USA, 978-1-4503-0700-0/11/06, June, 2011.
[11] James Abello, Yannis Kotidis, “Hierarchical graph indexing”, ACM, 1581137230/03/0011, Nov 2003.
[12] Maayan Geffet, Dror G. Feitelson, “Hierarchical indexing and document matching in BoW”, ACM 1-58113-345-6/01/0006, June 2001.
[13] H. V. Jagadish, Beng Chin Ooi, Kian-Lee Tan, Cui Yu, Rui Zhang, “iDistance: an adaptive B+-tree based indexing method for nearest neighbor search”, ACM Transactions on Database Systems, Vol. 30, No. 2, June 2005.
[14] Paolo Ferragina, Giovanni Manzini, “Indexing compressed text”, ACM, Vol. 52, No. 4, July 2005.
[15] Rushdi Shams, Robert E. Mercer, “Investigating keyphrase indexing with text Denoising”, ACM 978-1-4503-1154-0/12/06, June 2012.
[16] Jose E. Moreira, Dilma Da Silva, Parijat Dube, Maged M. Michael, Doron Shiloach, Li Zhang, “Scalability of nutch search engine”, ACM 978-1-59593-768-1/07/0006, June 2007.

Chetan P. Khilosiya, has done his BE-IT from University of Pune, and currently doing his ME-Computer at PICT, University of Pune. His area of interest include Information Retrieval, Networking and Data Mining.

Hemlata P. Channe has done her ME-Computer from Mumbai University. Her area of interest includes Computer Networks, Network Security and Distributed Computing.