XSearch: Benchmark the state-of-the-art Distributed Search Platforms (on disk)

Nilesh Jorwar, **Ashish Ryot**

njorwar@hawk.iit.edu, aryot@hawk.iit.edu

Abstract

Continuous growth in scientific and unstructured data makes it necessary to bring the parallel and distributed search platform into work to perform indexing. Each day as the size of data grows exponentially, so does the demand for the faster information retrieval thus necessitating to build the robust and faster search engines. While currently available search engines perform better on distributed file systems and well-structured data, they fail to provide timely response and support for complex and lengthy search queries coming from the users for the retrieval of data generated by HPCs for scientific purposes. This project involves implementation of Lucene and LucenePlusPlus libraries with parameter tuning to gain higher indexing and searching capability in multithreaded distributed environment. We thus tend to reduce the computation time needed for improving indexing throughput of the search platforms implemented on both single-node and multi-node clusters. Impact of storing the indexes on both the memory and disk will be compared against the parallel file systems by measuring the bottlenecks.

Keywords- **distributed filesystems, indexing, information retrieval, unstructured data.**

1. INTRODUCTION

Various large scale distributed and parallel file systems are becoming the data management infrastructures in big data and scientific computing environments. Grid systems, supercomputers and cloud systems are popular platforms that are pushing forward the boundaries in science sector to process and analyze the data distributed across clusters. Tools and applications running on such platforms generate hundreds of terabytes of data daily which needs to process and analyze. The use of computing centers and data centers built are handling the process of such large quantities of data by employing the parallel and distributed file systems such as Hadoop file System, OrangeFS, and Ceph.

The handling of vast amount of data goes through various phases of storage, access and process in more efficient manner and reasonable amount of time. The processing of such raw and scientific data is achieved using distributed search platforms that allows scientists and engineers to do search and extract relevant information from large scale storage systems in a short period of time, without no interference with other applications, also allows the complex queries submission and giving control over the data back to the users.

Distributed search platforms such as Apache Solr and Elasticsearch are prominent and provides distributed and multitenant-capable full-text search engine along with real-time indexing and dynamic clustering. Handling massive amount of unstructured scientific data of various formats necessitates high search capabilities that is provided by these search platforms.

2. RELATED WORK

Various studies and researches are going on the in the field of distributed search platforms indexing improvement. Distributed indexing platforms available these days allow search and extract information from large scale systems but lacks in faster information retrieval. It becomes need to have faster indexing in short amount of time with submission of complex queries and without interference of external applications. One of the research paper, presents search engines available, a slow search, primarily deals with high quality search experience while relaxing speed requirements [1]. Although this search engine gives us new search experience with high quality result but does not focus on the time factor considering the vast amount of data to search for indexing.

There is another paper which provides an alternative solution, the use of dynamic indexing in search engine, where indexer facilitates data organization for information retrieval and minimizes the time of query [2]. Indexing process is distributed over the cluster of computers in grid computing to improve the performance through distributed load. Although the process of distributing the indexed data over distributed memory helps faster indexing for small amount of data, say few petabytes of files, but significantly reduces the chances of faster data retrieval considering the millions of data resided on distributed storage systems. Additionally, this search engine results in overhead of maintaining continuous index updates from crawled document to the master node in distributed system.

3. PROPOSED SOLUTION

To perform the benchmarking of state-of-art distributed search platforms on disk, involves implementing Lucene and LucenePlusPlus libraries on single and multimode systems.

a. Lucene Library

The Lucene library is written in Java which gives us higher indexing and faster search time when tuned with different parameters. The implementation of this library involves storing the indexes on both the memory and disk. Its results are presented in the evaluation section. The tuning parameters used in indexing improvements are setting the Compound File Format to false which significantly resulted in reduction in index building time. For in-memory benchmarking, we set the RAM buffer size to 15 times less than the available Virtual Memory size which significantly resulted into faster search time since the larger indexes are stored on memory. Use of multithreading for both the files reading and index writing greatly improves indexing throughput. The architectural diagram i.e. Fig.1 shows structural representation of Lucene and LucenePlusPlus libraries.

b. LucenePlusPlus Library

The LucenePlusPlus library is written in C++ which results in less indexing and search throughput as compared to Lucene. This library is implemented on disk for its benchmarking and resulted in higher index size for metadata dumps.

c. Apache Solr Cloud with Zookeeper

Zookeeper library is setup over the 3 slaves and 1 master to achieve multi-node indexing efficiency.

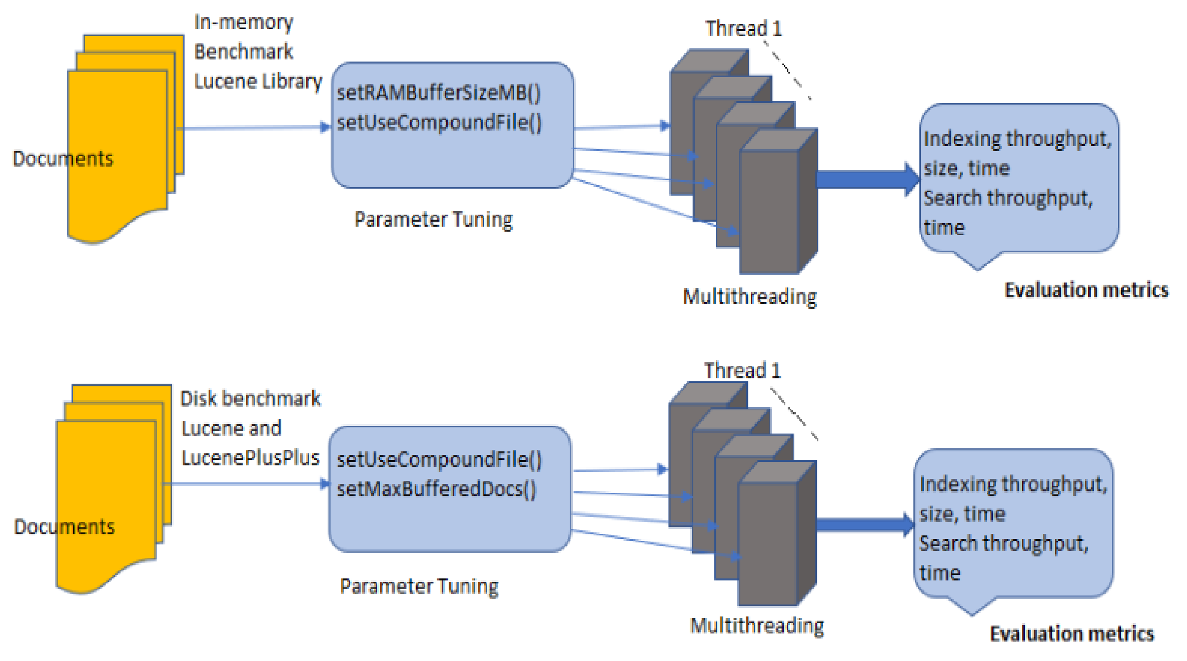


Fig. 1 Lucene and LucenePlusPlus Implementation

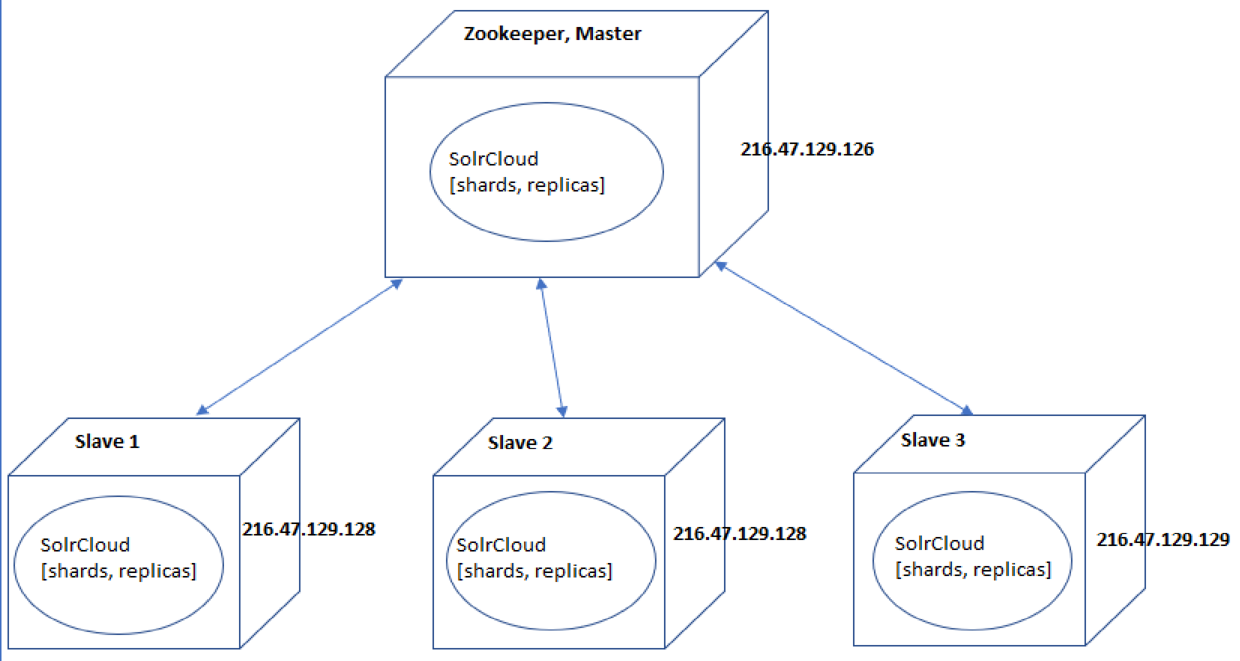


Fig. 2 Apache SolrCloud with Zookeeper

4. EVALUATION

Here we will discuss the experimental environment setup including the hardware and the software used, the various metrics that were used for the measurement of the performance of our evaluation on different workloads. We will also show and discuss the results that we got by deploying our implementation on the Cloud and other systems.

Testbed

The experiments discussed in this paper were conducted on the Chameleon Cloud testbed. Compute-Skylake bare metal nodes were used for running the experiments.

Number of CPUs: 2, Number of Threads: 48, RAM: 192GB, Storage: 240GB SSD with sequential read throughput of ~600MB/sec.

A few of the experiments involving multinode clusters and HDFS setup were performed on another system with the following configuration.

Virtual Machine:

CPU: Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz, Cores: 16

RAM: 16GB, Storage: 230GB

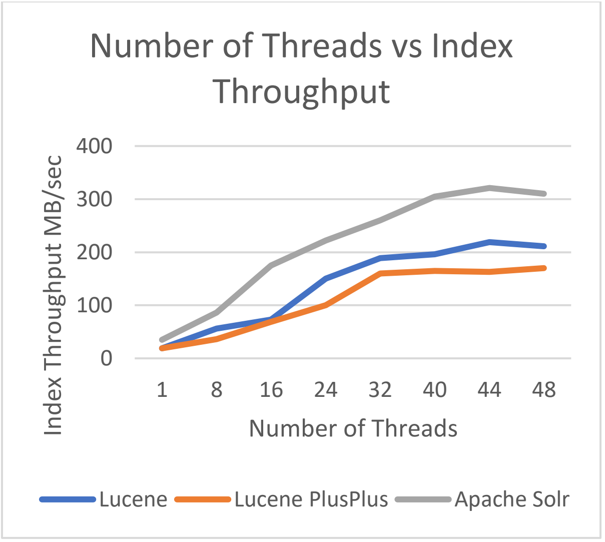
We used 4 of these VMs to setup the Master and slave nodes.

Experiments:

All the experiments were conducted on metadata dumps of size 180GB divided into smaller chunks of files of size ~4MB or ~17000 lines.

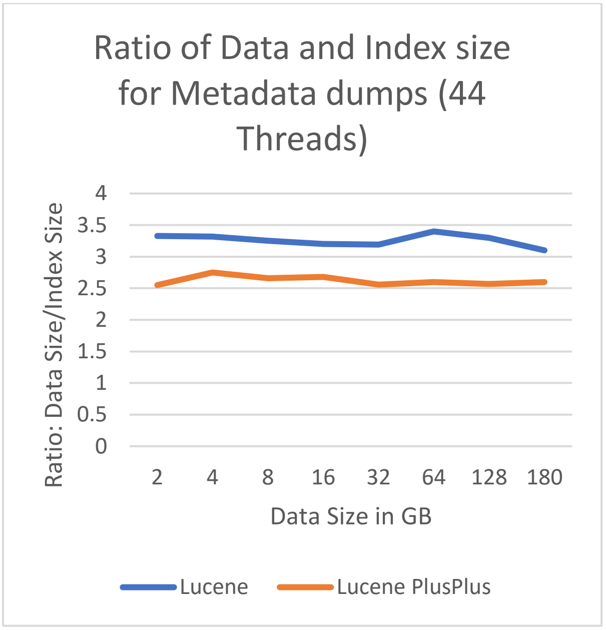
The experiments were also conducted on the text data dumps i.e. wiki dumps

a) **Indexing Throughput for different number of threads.**
We implemented our code using Lucene and Lucene PlusPlus libraries on varying number of threads like 1, 2, 4, 8, 16, 24, etc. and recorded our readings to see the effect of the increasing number of threads on the indexing throughput. We found out that for number of threads less than 10, our indexing code gave a low indexing throughput. As we increased the number of threads in subsequent experiments, we found an increase in the indexing throughput. We also found that on average the indexing throughput for Lucene PlusPlus is less than that of Apache Lucene. We also ran Apache Solr on the Skylake bare metal node and found out that Solr also follows the general trens of the Lucene and Lucene PlusPlus, i.e. as the multithreading increases, the indexing throughput of Solr also increases.
This increase in the throughput with the increasing number of threads is because of the reason that the file reading, writing and indexing work is divided between the threads equally and they perform the indexing on different parts of the file in parallel. We wrote code to implement strong scaling with the increasing multithreading and the workload.



b) Index file size comparison for Metadata dumps.

We ran multiple experiments using different number of threads on varying metadata dump size. We found that different number of threads did have any large effect on the index size. We also found that as the dump size increases, the index size also increases. In general, the size of the index formed by Lucene plusPlus was greater than the size of the index formed by Lucene. We also observed that the ratio of the Datasize and the index size stays almost constant as the data size increases.



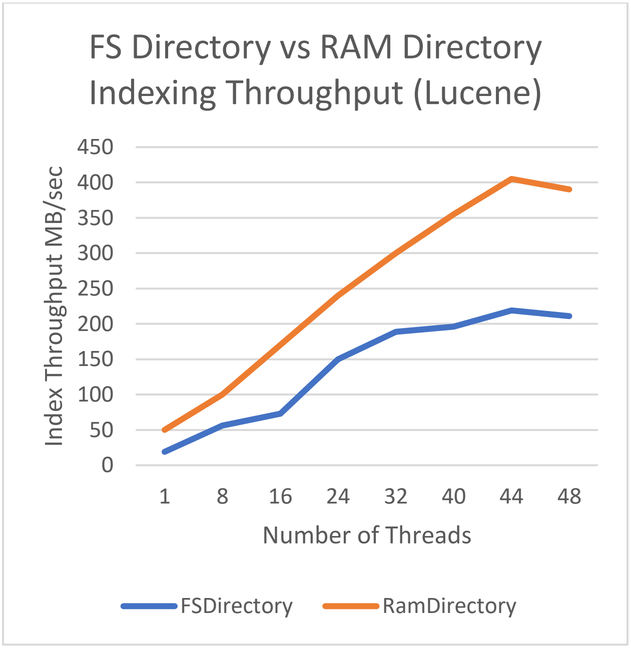
c) Comparison between Index stored on disk vs Index stored in Memory.

In the initial stages of the project we ran our Lucene and LucenePlusPlus implementation for searching and indexing in a way so that our index would be stored in the memory. Later we ran all our experiments to store the index on disk. On comparing the results we found out that the Memory implementation of the code gave very high indexing throughput as compared to the disk implementation. Also, as the number of threads increased, the indexing throughput increased for both the implementations.

This increased throughput for memory implementation is because of the fact that the threads have to write the index in memory which has a higher I/O throughput as compared to disk.

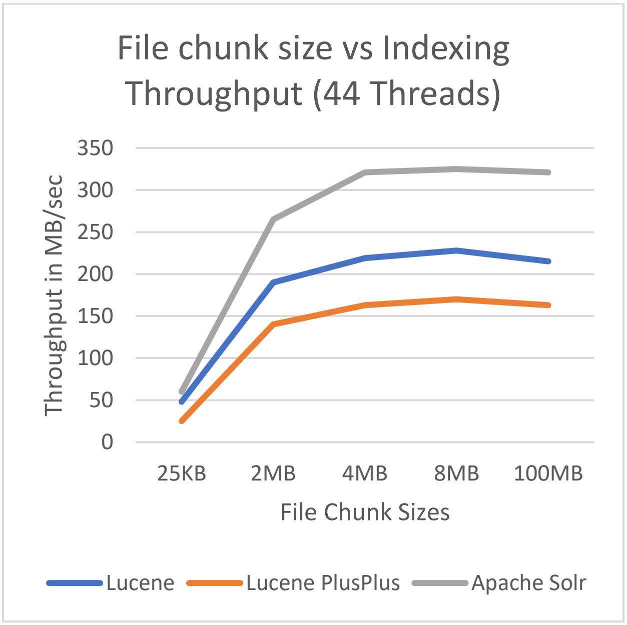
Although, the Memory implementation of the Indexing has a major drawback, i.e. as the size of the index increases for very large datasets, or workloads, the Memory will start filling up with index and the system will eventually run out of memory, causing the program to terminate with an error.

Therefore, Memory indexing is good if we want fast results on smaller datasets, but for really large workloads that exist in the real world it is better to use implementation of Indexing that stores it in disk.



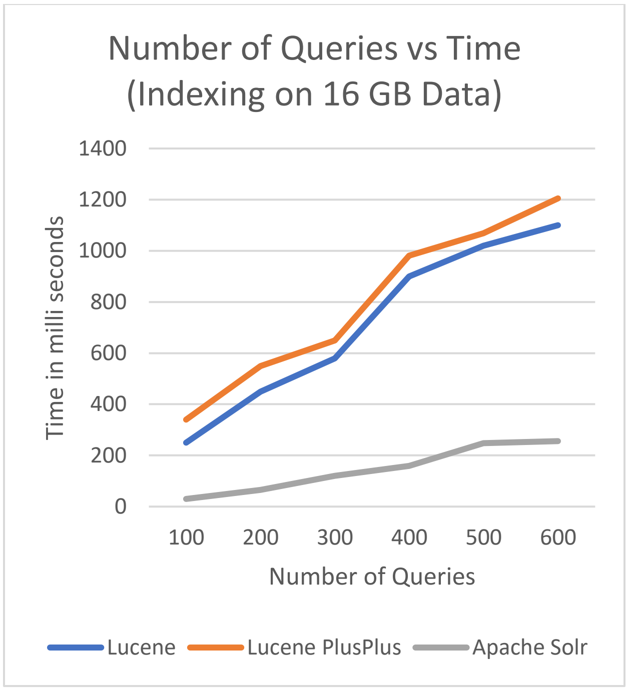
d) Effect of File Chunk size on Throughput:

We conducted most of our experiments on metadata dumps divided into small files of ~4MB each. But to decide on the appropriate chunk size we also conducted experiments on different file chunk sizes. In the graph below we can see that although the Indexing was Multithreaded, but it had little to no effect on the indexing throughput when the file chunk size was around 25KB. Even with 44 threads the program worked like we were performing indexing using a single thread. This could be because all of the threads were spending a lot of time in opening the file reading it and closing it and the multithreaded approach to reading and indexing it. 25 KB is much less than the appropriate block size to read files. We Found out that the file blocks of size greater than 3 MB and upto 100 MB performed well.



e) Query Throughput.

While running the above experiments we also measured the number of queries our code was able to perform on the index. The graph below shows us that as the number of queries increase the time then by the different implementation also increase. We got the best result for querying for Apache Solr. Our Model also followed the similar trend as followed by Solr.



f) Indexing Documents per Second for varying number of nodes (multi-node system)

We conducted this experiment on 3 slave nodes and one master as zookeeper to benchmark the Apache SolrCloud with Zookeeper.

We could see that 20000 documents are indexed per second of size 1KB with total time taken of 270 sec.

5. CONCLUSION AND FUTURE WORK

This paper presents the benchmarking of popular distributed search platforms such as Apache Solr and Elastricsearch to address the issues of searching and indexing the unstructured scientific data over distributed file systems. Parameters tuning of both these platforms (Lucene and LucenePlusPlus) along with single-node and multimode cluster setup helped improving indexing and searching capabilities significantly over wikipedia dumps and file systems metadata which can be seen from the results shown in evaluation stage. Multi-node cluster setup of using Zookeeper along with Apache Solr Cloud improved search efficiency over distributed systems. However, we could not implement ElasticSearch over multimode cluster environment which we plan to implement and evaluate in future to further improve indexing and searching efficiency along with quality search result.

6. ACKNOWLEDGEMENTS

We express our gratitude towards Dr. Ioan Raicu (Associate Professor- Illinois Institute of Technology) and Alexandru Iulian Orhean (Teaching Assistant) who provided us the support and guidance to benchmark parallel distributed search platforms over single and multimode systems.

REFERENCES

[1] Jaime Teevan, Kevyn Collins-Thompson, Ryen W. White, Susan T. Dumais & Yubin Kim, "Slow Search: Information Retrieval without Time Constraints" in Proceedings of HCIR 2013.

[2] M. E. Elaraby, Omaima Nomir, M. Sakre, "Dynamic and Distributed Indexing Architecture in Search Engine using Grid Computing" in International Journal of Computer Applications (0975 – 8887), Volume 55– No.5, October 2012.

[3] Lei Xu, Ziling Huang, Hong Jiang, Lei Tian, David Swanson, "VSFS: A Searchable Distributed File System" in Proceeding PDSW '14 Proceedings of the 9th Parallel Data Storage Workshop Pages 25-30

[4] S. Brin and L. Page, "Reprint of: The anatomy of a large-scale hypertextual web search engine," Computer networks, vol. 56, no. 18, pp. 3825–3833, 2012.

[5] Alexandru Iulian Orhean, Kyle Chard, Ioan Raicu, "XSearch: Distributed Information Retrieval in Large-Scale Storage Systems" Illinois Institute of Technology, 2018