A Detailed Study of Distributed Indexed Search Techniques using SOLR

B Jagadeeswar

Abstract: ***For any web application running on RDBMS databases as the backend, it might be a huge performance impact if a search needs to be performed on a table with millions of rows or if a query needs to be executed which joins multiple tables. In general, such kind of backend services make the website extremely slow. Document based reverse indexing can be a useful solution in these cases. SOLR is a standalone enterprise search server with a REST-like API. It has major features which include powerful full-text search, hit highlighting, faceted search, near real-time indexing, dynamic clustering, database integration, NoSQL features and rich document (e.g., Word, PDF and more) parsing, geospatial search, Security built in. Databases and SOLR have complementary strengths and weaknesses. SQL supports very simple wildcard-based text search with some simple normalization like matching upper case to lowercase. The problem is that these are full table scans. In SOLR all searchable words are stored in an "inverse index based", which searches orders of magnitude faster. However, designing this framework is quite challenging. This paper discusses the techniques that are highly reliable, scalable and fault tolerant which can help in setting up the distributed indexing, replication and load-balanced querying with a centralized configuration.***

Keywords: ***Reverse indexing, SOLR, RDBMS, Micro services, DB transactions***

I. INTRODUCTION

Web applications have advanced a lot since the days of client server architecture. Not long ago, all of the web-based application code was written and deployed as a single unit on the enterprise application servers [1] [2]. The database was the only separate unit in the deployment structure. Now modern frameworks for web design have come into place with the evolution of public cloud platforms. This revolution has certainly improved the network latency and performance of the applications. However, the biggest challenge of improving the performance of web pages cannot be addressed without the inclusion of document based reverse index solutions like SOLR in the architecture, when dealing with large RDBMS databases. This paper provides the introduction and journey of SOLR as a prominent reverse index solution. The initial version of SOLR used Master Slave architecture wherein there is a master for performing all the write operations and there is a reader to perform all the read operations. This architecture posed lot of challenges: you must use all your read and write operations in a stagnated manner. It can be used only for limited data sets and it is not capable for scaling to huge data sets. The newer versions of SOLR was redesigned to handle huge data sets.

Revised Manuscript Received on August 15, 2020.
* Correspondence Author

B Jagadeeswar*, Department – Computer Science, Vijayam Degree & PG College, Chittoor, Sri Venkateshwara University, (A.P.), India.

Currently, there are large number of websites with large amount of data available, which is necessary to handle in an efficient way in modern era.

- The data is being generated within organizations, either as output or intermediate process of production systems or by digitizing existing documents.
- The popularity of content management systems (CMS, Content Management Systems) as portals general and as platforms for collaboration.
- The Web 2.0, roughly defined as the current set of applications with high levels of interaction and access to multimedia data.

In summary, there has been an exponential growth in volumes of information produced which, in turn, implies handling terabytes and petabytes of information instead of gigabytes. This scenario has led to the challenge of improving the information retrieval search tools using different/new techniques. Scalability, availability, and performance in handling large volumes of information are now mandatory for most applications in this context, usually requiring techniques of distributed systems. Some of the techniques presented in this work include load balancing, replication, and horizontal distribution (sharding) of information. For instance, The White House has used a combination of Drupal and Apache SOLR in its portal of document access/contents. In general, solutions to this problem must include strategies for scalability, availability, and performance. The block diagram given below shows the high-level architecture for SOLR based indexing as shown in Figure 1.

II. BACKGROUND ABOUT INDEXING TECHNOLOGY

Sequential Search was the old traditional way of searching for the documents due to its scalability. Some modern data structures are needed to overcome this problem. Indexing provides data search and pulling the data very rapidly and accurately. Indexing process includes analysis, Processing of documents, tokenization, phone analytics, geo spatial searching, etc. This document explains how feasible the solution is for Performance and heterogeneous data, fault tolerance, automatic fail-over, different hardware problems.

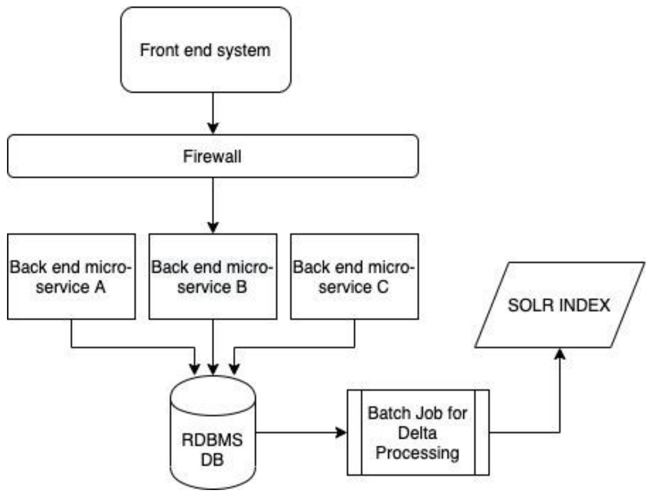

Figure I - High level SOLR integration design.

A. Performance and heterogenous data

Relational or SQL databases have many issues while storing the information in relational tables. As the data is not just the information in the database, but instead it has lot of documents, images, videos, audios, emails, etc. This lead to increase the size of the data that is stored. It needs most of the time re-structuring before the information is being pulled from the database.

B. Fault Tolerance

One of the important part that needs to be considered in the distributed system while fetching the data is Fault tolerance. Data that we fetched must be consistent across when sometimes one or two instances went down or not reachable. This paper concentrates on maintain the consistency, how we store the data in distributed environment and recovery.

C. Heterogeneous Hardware Platforms

To achieve the best performance in data fetching is possible when we increase the number of instances when we are performing the search. NoSQL provides solution for maintaining huge volumes of data is depending on the diverse computing nodes. There are n number of approaches for vast environments which are discussed in the paper.

III. PROPOSED METHODOLOGY

There are many common approaches used in the current No SQL databases which includes Shard based Indexing, Replicating the data for load balancing, Map/reduce processing [6]. We can achieve this using Solr Cloud which intern uses Apache Zookeeper. As a part of this research we have experimented with an RDBMS database [8] (Postgres) which contains few tables with a large data set. We indexed this using a full index on SOLR software running on odd number of nodes. We started with 1 node and repeated the experiment with 3 nodes. In the first run we used SOLR alone as shown in Figure III and in the second run we used it along with Zookeeper as shown in Figure IV. In this run, all these nodes are governed by a software called Zookeeper which runs on an external server node. In the following sections we explain the various techniques which can be used in the process of indexing. As explained below - indexing on shards, shared nothing, Scatter and Gather on distributed data and Map reduce are the 4 major techniques to be followed in the reverse indexing solutions. For our experiment we have used indexing on shards and shared nothing techniques. In these cases, a caller who makes the query is unaware of how the indexed data is distributed in the backend nodes. We experimented by distributing the indexes directly on SOLR and checked how the system performed on the queries which cause heavy load. In the second run, multiple nodes for SOLR were used. A servicing software like Zookeeper is introduced which takes care of the health of each node containing the indexed data and also does the routing, querying and returning of the results to the caller. We compared how the system performs in both the cases. The below section provides the details.

A. Indexing on Shards

Sharding is a horizontal partitioning of information in a structured database. It provides various capabilities for scaling, granting to divide information and indexes on multiple servers are called as Shards. Indexing Shards is the technique of generating a data structure that expedite the searching and retrieving data in its original output. Each search query that we are performing will be processed in each Shard and at last the response will be aggregated. This approach is very helpful when we have the large volume of data.

B. Shared Nothing Data Distribution

Shared nothing [7] focuses independence of nodes, distribution of information and processing. A shard is a shared nothing node which handles a set of documents indexed by any criteria. Also, a shard has its own mechanisms for ranking, sorting, and retrieval of information, depending on information or application needs. In all cases, techniques can be combined with traditional databases, such as replication and parallelization on shared disk (a traditional cluster with a storage area network). Also, these distributions make easier using independent heterogeneous nodes with their own memory unit, disk storage and processing. Data replication for load balancing, scatter and gather on distributed data, and map/reduce processing are some techniques used for coordinating the shared nothing nodes.

C. Scatter and Gather on Distributed Data

This type of approach is used when we have the data is not replicated and the search phrase in coming from a person to each instance known to have the data. Once we get that each node processes and sends back the response to the info was found in its memory. All responses are processed and consolidates into one unique reply to the requested source. One of the added advantages of this approach is distribution of the data from one instance to the other one. An ordered distribution is created, and which won't be visible to the overall coordinator. Here we do have disadvantages. Some logical partition(s) requires knowledge and information regarding the data that must be saved. In these scenarios we can use Map/reduce approach efficiently for this problem.

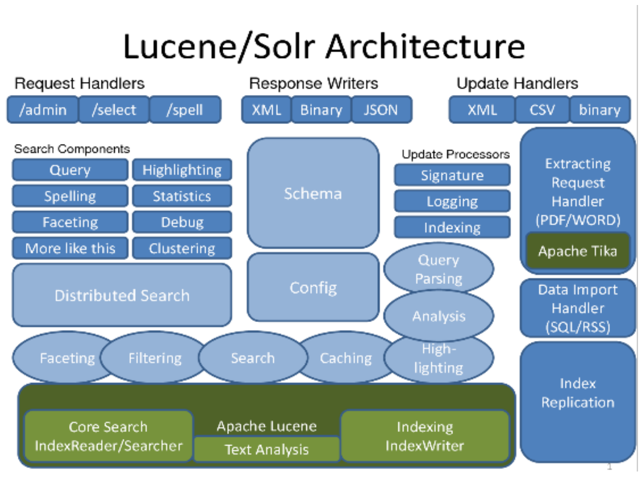

Figure II – Lucene SOLR architecture.

D. Map/Reduce Processing

It is a program model for distributed computing on large data sets. This is an algorithm contains two distinct tasks – Map and Reduce. First most we read the block of data and gives key-value pairs as output. The output is going to be an input to the Reducer. Reducer aggregates the key-value pairs into smaller set of key-value pairs and sends as a final output. It takes the advantage of the locality of data. This model provides a versatile way for partitioning data which in-turn to be a smart distribute on self-contained various instances. One more added advantage is saving storage space within a document based on the result of shared keys by reducing them.

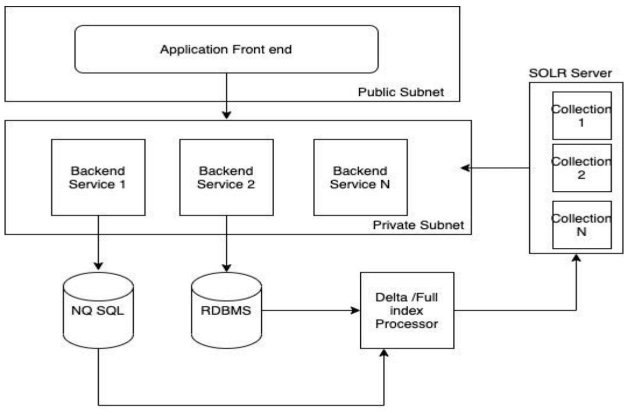

Figure III– Design with SOLR on one node.

IV. RESULT ANALYSIS

We conducted an experiment by taking a Node.JS microservice based backend and Angular.JS based front end which runs on the Postgres as the database layer. We created a sample table which contains more than 10 million records and performed the queries. Let us call this entity E1. This table joins with multiple child entities using the foreign key relationships. Let us call them entities E2 and E3 respectively. When we ran a select query on the database alone by joining E1,E2,E3 using the left outer joins it took 12 seconds to display the data in the paginated grid on the Angular JS front end. When we tried to add the result facets by grouping the results, it nearly took 21 seconds to display the page. When we repeated the experiment by indexing all the data in SOLR and had the Node.JS microservice interact with SOLR rather than the database directly. When a query for the data is done from the front end, it took 8 seconds to get the results back. When we introduced the facets to this run, it took 10 seconds to display the result. We found that faceting runs much faster when it is indexed. However, when we increased the number of records by adding 100 million more records, the SOLR search was taking more time. This was because of the number of indexed documents were more. To handle this situation, we repeated the experiment by using multiple SOLR nodes in the backend. We used 3 nodes for this experiment. We also used Zookeeper on a different node which manages this SOLR ensemble. Now with this setup the page started to load much faster for the same query with the increased data volume. We observed that the page loads the grid with the paginated results along with the faceted result set in less than 5 seconds. We observed that reverse indexing with a single node brought in nearly 110% of performance gain. But it suffered as the data volume increased. When Zookeeper ensemble was introduced and the indexes were distributed on multiple shards in the backend, it brought in more than 300% gain in the performance.

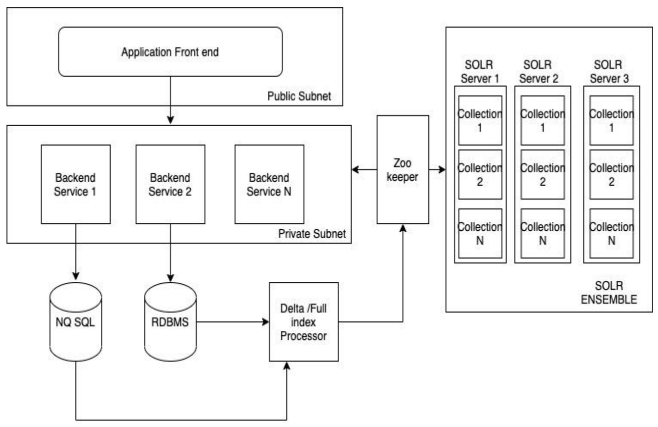

Figure IV– Design with SOLR ensemble and Zookeeper.

V. CONCLUSION

In the modern distributed architecture with cloud-based systems the data volumes are substantially high [9]. Persisting such data and querying it on plain RDBMS systems slows down the performance and increases the page load times. Adding the reverse indexing mechanisms like SOLR would significantly improve the performance and page load times. But they suffer when the number of joins increase, or number of data records increase beyond a certain limit. In these cases, using distributed shards and having a governing layer like Zookeeper greatly improves the performance. This we have conclusively proved using the experiment we conducted. Further research needs to be done on Map Reduce algorithms.

REFERENCES

1. Salah, Tasneem & Zemerly, Jamal & Yeob Yeun, Chan & Al-Qutayri, Mahmoud & Al-Hammadi, Yousof. (2016). The evolution of distributed systems towards microservices architecture. 318-325. 10.1109/ICITST.2016.7856721
2. Chaitanya K Rudrabhatla. A Systematic Study of Micro Service Architecture Evolution and their Deployment Patterns. International Journal of Computer Applications 182(29):18-24, November 2018
3. J. Chris Anderson, Jan Lehnardt, Noah Slater, CouchDB: The Definitive Guide, O'Reilly Media, Jan. 2010, ISBN 1449379680.
4. Jeffrey Dean, Sanjay Ghemawat, “MapReduce: Simplied Data Processing on Large Clusters,” Communica1 Server S1 R1 S2 R2 tions of the ACM - 50th anniversary issue: 1958 - 2008, Vol. 51, Issue 1, Jan. 2008.

5. Erik Hatcher, Otis Gospodnetić, Lucene in Action, 2nd. ed, Manning Publications Co., 2004.
6. Ken North, "The NoSQL Alternative, Low-cost, highperformance database options make gains," Information Week, May 2010.
7. Michael Stonebraker, "The Case for Shared Nothing", Database Engineering, Vol. 9, No. 1, 1986, http://db.cs. berkeley. edu/papers/hpts85- nothing.pdf
8. ACID (Atomicity, Consistency, Isolation, and Durability.),in Dictionary of E-Business, Hoboken, Wiley, 2003.
9. Nasser Thabet e Tariq Rahim Soomro, Big Data Challenges, in Journal of Computer Engineering & Information Technology}, 2015, DOI:10.4172/2324-9307.1000133.
10. P. Zikopoulos, C. Eaton, Understanding Big Data: Analytics for Enterprise Class Hadoop and Streaming Data, McGraw-Hill Education, 2011

AUTHOR PROFILE

B. Jagadeeswar works as the Head of the Department in Computer Science department at Vijayam Degree & PG College, Sri Venkateshwara University, Chittoor, AP. He received master's degree in Computer science from SK University, Anantapur, AP in the year 2000.