Sourabh Sethi. 2023, 11:6
ISSN (Online): 2348-4098
ISSN (Print): 2395-4752

International Journal of Science, Engineering and Technology

An Open Access Journal

Analyzing and Designing a Full-Text Enterprise Search Engine for Data-Intensive Applications

Sourabh Sethi, Sarah Panda, Ravi Karmuru, Tarun Tayal
Department of Computer Science & Information Systems
Birla Institute of Technology and Science, Pilani, Rajasthan, India

Abstract- The process of designing and constructing an Enterprise Search Engine comes with numerous challenges. One major hurdle is creating a search feature capable of efficiently navigating through an extensive volume of generated data, a task that SQL databases have struggled with over the past decade. In SQL, the conventional approach involves constructing a B+ tree encompassing all posts. However, this method is effective primarily when dealing with smaller posts (single or double words). For larger text, the need to traverse each tree node becomes time-consuming. NoSQL databases encounter a similar issue; in key-value or column-family stores, searching for matching values or columns involves navigating through every row, a process that can be quite prolonged. The same holds true for document stores. In this research article, we delve into the challenges associated with implementing an enterprise search engine and propose potential solutions.

Keywords- Elastic search Stack, Apache Solr Search Stack, Apache Lucene, DXPs, and Digital Experience Platforms.

I. INTRODUCTION

Within the SQL framework, we typically construct a B+ tree encompassing all posts. This method is effective for smaller posts, typically comprised of single words or two words. However, for larger text, the necessity to navigate through every tree node becomes time-consuming. For instance, if someone searches for "job posting," seeking all posts containing either "job posting" or "job post" in their body, the query would look something like the following:

SELECT * FROM posts WHERE content LIKE "%job post%".

The provided query would result in a complete table scan, necessitating the examination of every row and conducting a string search within each post content. With N rows and M characters, the time complexity becomes O(N*M^2). This is not an optimal scenario and could potentially lead to performance issues, possibly causing a system like LinkedIn to experience downtime.

1. Issue with No SQL Databases

Encountering a comparable challenge would occur when employing a NoSQL database. In a key-value store or column-family store, one would need to traverse through each row and search for matching values or columns, a process that could be extremely time-consuming. The same issue applies to a document store.

II. PROPOSED DESIGN

To address this issue, we can employ the following data structures: Hashmap or Trie. In the case of a Hashmap, the key-value pairs can be structured as follows: the key represents the word to be searched, and the value contains a list of document (or review) IDs where the queried word is present. This arrangement is commonly known as an INVERSE INDEX. [1] APACHE LUCENE implements a similar concept, where each entry, such as an entire post, is referred to as a document. The following steps are involved in this process:

The initial step involves Character Elimination, where characters such as "a," "an," "the," etc., are removed. Despite its name, this phase also encompasses word elimination. The second step is Tokenization, wherein the entire post is segmented into individual words. The third step, Token Indexing, breaks down all tokens into their root words, a process also known as stemming. For instance, the words "ran" and "running" are reduced to the root word "run," and "crashed" and "crashes" are stemmed to "crash." Moving on to the fourth step, Reverse Indexing is performed. In this phase, we store the (document id, position) pair for each word. For example, if after the third phase the indexed words for document 5 are: "decent - 1," "product - 2," "wrote - 3," "money - 4," then in the reverse indexing phase, the word "decent" would be associated with a list like [(5,1)], where each element of the list represents a pair of (document id, position id).

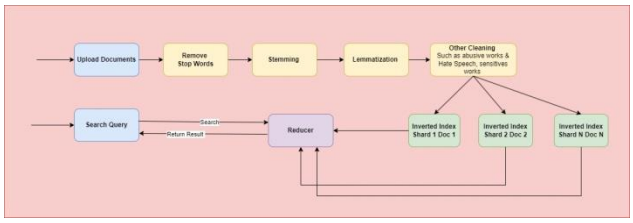

Figure 1: Upload/Query

1. Use Cases of Full Text Search

Applications of full-text search include: processing logs, indexing user-entered text, indexing text files/documents (e.g., resume indexing for searching based on resume content), and site indexing.

III. ELASTIC SEARCH

Apache Lucene is impressive, but it operates as software designed for a single machine. However, relying on a single machine presents challenges such as becoming a single point of failure, potential limitations in storage capacity for documents, and the inability to efficiently manage high volumes of traffic. Consequently, Elastic Search was developed on top of Lucene to address these scalability issues.

When considering Elastic Search's priorities, the question arises: should it prioritize greater availability or stronger consistency? In the case of search systems, such as LinkedIn's post search, a high level of consistency is not always a strict requirement. Therefore, Elastic Search may lean towards prioritizing high availability. Let's delve into the terminology encompassing Document, Index, and Node. A Document represents an entity containing text for indexing, like an entire LinkedIn post. An Index is a compilation of indexed documents; for example, LinkedIn posts could constitute one index, while resumes might form a distinct index. A Node, in this context, signifies a physical or virtual machine.

1. Sharding

If there are an overwhelming number of documents, making it impractical to fit the entire dataset on a single machine, how would you approach sharding?

Elastic search employs sharding based on document ID, ensuring that a document, identified by its document ID, is never divided across multiple shards; instead, it exclusively belongs to a specific shard. The sharding algorithm entails specifying the desired number of shards during Elastic search setup. If the number of shards is either fixed or infrequently modified, a simpler approach than consistent hashing is employed: a document with a document ID is assigned to shard number (hash (document_ id) % number of shards).

2. Replication

Similar to specifying the number of shards, you can also define the number of replicas during the setup phase. Replicas are essential because machines can fail, and having replicas ensures that even in the event of machine failures, the shard remains active, and data is preserved. Additionally, a higher number of replicas aids in distributing the load of read operations, as a read can be directed to any of the replicas. Just as in the master-slave model, one of the replicas within the shard is designated as the primary/master, while the remaining replicas function as followers/slaves. For instance, if num_ nodes = 3, num_ shards = 2 (0 and 1), and num _replicas = 3, the configuration might look like the following in figure 2.

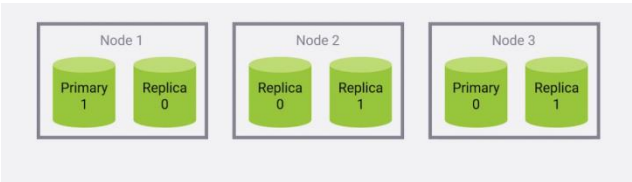

Figure 2: Replicarion

When there are fewer nodes, it results in multiple shards coexisting on the same node. This can be

mitigated by introducing additional nodes into the cluster. The increased number of nodes in the cluster also provides the flexibility to configure and manage the allocation of shards per node.

IV. READ AND WRITE FLOW

Write (Index a new document): To index a new document, the system locates the appropriate shard for the document_ id and identifies the node containing the primary replica. The request to index the document, similar to Lucene's write process described earlier, is then transmitted to that node (primary replica). Updates from the primary replica are asynchronously propagated to the slave replicas. Read (Given a phrase, find matching documents along with matching positions): As documents are distributed across shards, and any document could potentially match the given phrase, reading in Elastic search involves reading in every shard. Upon receiving a read request, the responsible node forwards it to the nodes holding the relevant shards, gathers the responses, and communicates with the client. This node is termed the coordinating node for that particular request. The fundamental flow is as follows: Resolve the read requests to the relevant shards. Select an active copy of each relevant shard from the shard replication group, which can be either the primary or a replica. Dispatch shard-level read requests to the chosen copies. Aggregate the results and respond. In cases where a shard fails to respond to a read request, the coordinating node redirects the request to another shard copy within the same replication group. Repeated failures may lead to the unavailability of shard copies. For swift responses, certain Elastic search APIs provide partial results if one or more shards encounter failures.

V. CONCLUSION

Apache Solr boasts exceptional indexing and search speed, featuring a remarkably compact index size and impressive extensibility. It doubles as a versatile repository and incorporates various additional functions, including imprecise search capabilities and seamless scalability. However, it is worth noting that Solr operates as a Java server within a servlet container, functioning as a web service with XML/JSON/CSV interfaces. [5] On the other hand, Elastics earch, built on Apache Lucene, exhibits slightly lower indexing and searching speeds compared to Sphinx. Despite this, Elastic search offers a comprehensive suite of tools beyond search and storage, encompassing visualization, log collection, and encryption systems. Its scalability and ability to handle intricate data structures make it an ideal choice for analytical platforms. While Elastic search may not be the most user-friendly, it compensates with a plethora of advanced features. Notably, it consumes minimal memory, and its incremental indexing is as swift as indexing multiple documents concurrently.

ACKNOWLEDGEMENTS

We have collaborated with strategic partners whose invaluable contributions have been pivotal in identifying search solutions within the market. Their significant role extends to implementing diverse search solutions across a range of enterprise applications over the years.

REFERENCES

1. Gormley, Clinton, and Zachary Tong. Elastic search: the definitive guide: a distributed real-time search and analytics engine. " O'Reilly Media, Inc.", 2015.
2. Kononenko, Oleksii, Olga Baysal, Reid Holmes, and Michael W. Godfrey. "Mining modern repositories with elastic search." In Proceedings of the 11th working conference on mining software repositories, pp. 328-331. 2014.
3. Kuc, Rafal, and Marek Rogozinski. Elastic search server. Packt Publishing Ltd, 2013.
4. Amato, Giuseppe, Paolo Bolettieri, Fabio Carrara, Fabrizio Falchi, and Claudio Gennaro. "Large-scale image retrieval with elastic search." In The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, pp. 925-928. 2018.
5. Voit, Aleksei, Aleksei Stankus, Shamil Magomedov, and Irina Ivanova. "Big data processing for full-text search and visualization with Elastic search." International journal of advanced computer science and applications 8, no. 12 (2017).
6. Bagnasco, S., D. Berzano, A. Guarise, S. Lusso, M. Masera, and S. Vallero. "Monitoring of IaaS and scientific applications on the Cloud using the Elastic search ecosystem." In Journal of physics: Conference series, vol. 608, no. 1, p. 012016. IOP Publishing, 2015.
7. Betke, Eugen, and Julian Kunkel. "Real-time I/O-monitoring of HPC applications with SIOX, elastic

search, Grafana and FUSE." In High Performance Computing: ISC High Performance 2017 International Workshops, DRBSD, ExaComm, HCPM, HPC-IODC, IWOPH, IXPUG, P^ 3MA, VHPC, Visualization at Scale, WOPSSS, Frankfurt, Germany, June 18-22, 2017, Revised Selected Papers 32, pp. 174-186. Springer International Publishing, 2017.

8. Sholihah, Walidatush, Sangga Pripambudi, and Anggi Mardiyono. "Log event management server menggunakan elastic search logstash kibana (elk stack)." JTIM: Jurnal Teknologi Informasi dan Multimedia 2, no. 1 (2020): 12-20.
9. Bevendorff, Janek, Benno Stein, Matthias Hagen, and Martin Potthast. "Elastic chatnoir: Search engine for the clueweb and the common crawl." In Advances in Information Retrieval: 40th European Conference on IR Research, ECIR 2018, Grenoble, France, March 26-29, 2018, Proceedings 40, pp. 820-824. Springer International Publishing, 2018.
10. Shivakumar, Shailesh Kumar, and Sourabhh Sethii. Building digital experience platforms: A guide to developing next-generation enterprise applications. APress, 2019.
11. Sourabh Sethi , Sarah Panda . The Evolution of Monolithic DXPs to Micro service based DXPs. TechRxiv. October 18, 2023. DOI: 10.36227/techrxiv.24328504.v1

AUTHOR'S DETAIL

Sourabh Sethi is working as Independent Researcher and Technology Lead at Infosys Technologies Limited. He helps organizations to develop end-to-end digital solutions and adopt emerging technologies using agile methodology. He has achieved multiple honors, including "Most Valuable Player," "Insta Awards" from the heads of his unit at Infosys. He holds a master's degree in software systems, specialized in data analytics, from BITS Pilani, Rajasthan, India. Member of the IEEE Computer Society. Contact him at sourabhsethi@ieee.org.

Sarah Panda is working as Senior Applied Scientist at Microsoft, Research & Incubations in Seattle. In her current role, she works on end-to-end AI solutions to extract process & search information from different kind of data, like pdfs, structure content, text, tables and graphs. She graduated with a Master's degree from Columbia University in Machine Learning projects using NLP & Computer Vision in her time at Amazon and Microsoft. Contact her at sp3206@columbia.edu.

Ravi Kamuru is working as "Programmer Analyst / Developer" at New York Technology Partners. In his current role working on "End to End functional development to meet client needs by adapting to the latest technology development frameworks. Introducing "Client Customization Framework" which helps to solve generic issues of global clients. Received many awards like "Emerging Leader" and "Above and Beyond". Holds "Bachelor of Technology" from JNTU Hyderabad India. Contact him at kamururavi.eee230@gmail.com

Tarun Tayal currently serves as the Technology Lead at Infosys Technology Limited and is actively involved in technology-related research. His role involves working as a full-stack architect, specializing in Java, .NET, and Angular technologies. With a track record of delivering end-to-end solutions for numerous web applications across diverse clients, Tarun has received multiple accolades from clients and organizational awards for his significant contributions. He holds a bachelor's degree from Punjab Technical University and is certified by Microsoft in the .NET framework.