Optimizing Search in Un-Sharded Large-Scale Distributed Systems

Suraj Chafle[1], Jonathan Wu[2], Ioan Raicu[1], Kyle Chard[3]

Illinois Institute of Technology[1], Washington University in St. Louis[2], University of Chicago[3]

schafle@hawk.iit.edu, wu.jonathan@wustl.edu, iraicu@cs.iit.edu, chard@uchicago.edu

***Abstract*— Distributed file systems and storage networks are used to store large volumes of unstructured data. While these systems support large-scale storage, they create new challenges relating to efficiently discovering, accessing, managing, and analyzing distributed data. At the core of these challenges is the need to support scalable discovery of unstructured data. Traditional search methods leverage centralized and globally sharded indexes. We present a distributed search framework that does not rely on sharding and can be applied to a range of distributed storage models. Our approach is built on top of Lucene and utilizes search trees to distribute and parallelize queries. To further optimize query performance we explore methods to prioritize indexes based on size. We evaluate our search framework against alternatives, Grep and Solr, comparing our hierarchical query distribution with a centralized model. Our implementation proved to be faster and scale better.**

Keywords—Distributed Systems; search; unsharded; Lucene

I. Introduction

An estimated 70% of the world's data remains unstructured, with the value growing about 60% each year[3]. The move towards big data and computationally intensive data analytics algorithms has motivated widespread usage of systems capable of handling huge workloads and storing large amounts of data. Typically, such systems are distributed, rather than centralized, because distribution avoids the exponential increase of cost associated with single servers. Distributed storage systems may take the form of computers connected over the web or racks of servers in a supercomputer. While these systems improve performance and decrease costs they create new challenges with respect to common data management and analysis operations, such as data discovery.

Most traditional operating systems support search via implementation of query primitives (e.g., find or grep) or using a desktop search. Distributed implementations, such as Hadoop Grep can be used on specific distributed systems; however, they often require specialized software and are non-trivial to use. Many desktop search engines use pattern matching rather than indexing to perform deep search. However, in a large distributed environment such an operation is infeasible due to the expense and overhead required to perform queries over a large number of nodes.

Distributed systems and specifically distributed file systems have existed for over two decades. While there are software solutions for searching within distributed systems, to the best of our knowledge there is no framework for searching in an unsharded environment. Examples of such environments include the Fusion File System and Globus data-management service. Here we present a general search framework that can be applied to different distributed search scenarios.

II. Problem Statement

We aim to develop a distributed "desktop-like" search across unsharded distributed file and storage systems.

Search Our solution must include the four main parts of a search engine (indexer, query processer, searcher, and scorer) and must support free-text search with an emphasis on speed and scalability. It should support near-real-time discovery of documents on any node from any other node in the cluster.

Environment The unsharded nature of our environment means that documents are not split up between nodes, but rather each document remains intact on a single node and the index in which the document is registered is co-located with the document itself. Furthermore, we do not guarantee that the information stored in our system is balanced between all the nodes. The purpose of this separation is to keep individual nodes autonomous and minimize network traffic.

III. Archetechture and Implementation

We build our approach on Apache Lucene, the de facto standard large-scale indexing and query-processing engine. Lucene is a Java library on which many popular search engines like Solr and ElasticSearch are implemented. Our solution leverages a C++ version of Lucene to index documents, maintain search indexes, perform queries, and score results. We have developed a custom TCP/IP protocol for sending messages (e.g., queries and results) between nodes. Due to the unsharded nature of the environment, each node is responsible for maintaining an index of only the documents located on that node.

Lucene Lucene stores data in the form of *documents* where each document contains a number of *fields*. Fields are populated by tokenization of the text values of the document. To support partial and free-text search, the values stored in fields are matched against queries, then the results are scored and ranked, and finally the matching documents are returned. The number of relevant fields within a single document

NSF-1461260

influences the scoring of the document, meaning more relevant documents are given higher scores. Lucene provides near-real-time indexing-to-search capabilities.

Server-Client Model The search engine consists of a command line client interface and a server deployed on each node. When a query is submitted using the client (on any node), that client communicates to the server on that node, which begins searching the current node. The server then propogates the query to other nodes. After each node's server has executed the query, the results are ultimately argregated on the server of the originating node and sent back to the client.

Query Distribution After a server receives a query, its parent and children nodes are calculated using an ordered membership list of all other nodes. The membership list allows for dynamic changes in cluster membership; however, it also requires that nodes must update the list when joining or leaving the cluster. The distribution tree, which originates at the source node, includes all other nodes to distribute queries and collect results. This approach allows quesries to be distributed faster than a broadcast and reduces traffic at the client. When assembling results, each non-leaf node waits for and collects the results from each of its children. The results are combined with its own and returned to its parent. We futher optimized the query distribution phase by prioritizing nodes with larger indexes, which are more likely to require a longer searchtime.

IV. Evaluation

We evaluated our implementation on clusters of size 3, 7, 15 and 31 using m3.large instances on Amazon Web Services.

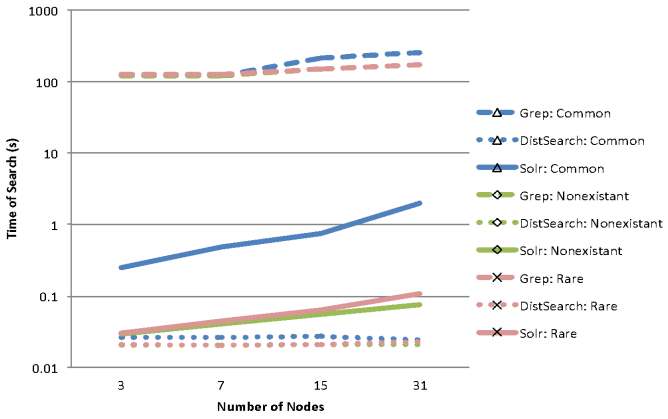

Fig. 1 Our implementation (DistSearch) vs Solr vs Grep

To simulate large amounts of textual data, we distributed an average of 90000 Wikipedia files across each node. We then explored the performance on a range of queries: common queries (>100 results), a nonexistant query (0 results), and a rare query (1-10 results). We compared performance of these queries using our implementation as well as distributed Grep and Apache Solr.

Our implementation outperformed Grep and Solr. Grep provides a comparison with a standard method for most "desktop-like" searches. As expected, the pattern matching used by Grep is significantly slower than than index look-ups. Solr is a more comparable alternative, since it is also built on Lucene and therefore utilizes indexes. The fact that Solr is writen in java and our implementation is writen in C++ may account for some of the observed overhead difference. On average, our implementation performed 5x, 12x, 16x and 21x better than Solr on 3, 7, 15 and 31 nodes respectively. We expect our implementation to continue to scale better than alternatives because of the underlying tree structure it is built on top of, which reduces query distribution and result collection bottlenecks.

V. Related Work

FusionFS[1] The Fusion File System includes a basic search model implemented using FusionFS services like caching and metadata management. The fundamental distribution pattern relies on a broadcast from the searching node, which is comparable to the "star" distribution pattern we evaluated against.

ElasticSearch [2] ElasticSearch is built on top of Solr and has been optimized in completely distributed environments that rely on sharding of documents and requires a complicated execution model. A search request must consult a copy of every shard in the indices of interest to see if they have matching documents. This implementation also uses a broadcast distribution.

VI. Summary and Future Work

We have developed a tree-based search framework for distributing queries in a distributed storage system. Our approach outperforms popular alternatives at clusters of all sizes.

Our future work focuses on integrating our general search framework with other environments. Specifically, we intend to integrate our framework with FusionFS and Globus. We also aim to improve the fault tolerance of our implementation and investigate methods for reducing search and wait times on each node via new distribution algorithms.

Acknowledgments

This work is supported in part by the Nation Science Foundation under awards NSF-1461260 (REU).

References

[1] Ijagbone, Itua. "Scalable Indexing and Searching on Distributed File Systems." Thesis. Illinois Institute of Technology, 2016. Print.

[2] ElasticSearch. Computer software. Elastic Stack and Product Documentation. Vers. 2.3. Elastic, n.d. Web.

[3] M. Wall, "Big Data: Are you ready for blast-off", BBC Business News, March 2014.

[4] Ian Foster. Globus Online: Accelerating and Democratizing Science through Cloud-Based Services, IEEE Internet Computingspacer.gif, 2011