JSS ACADEMY OF TECHNICAL EDUCATION

DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING

FILE STRUCTURES ACTIVITY REPORT
ON

Case Study: Analysis of Indexing and Searching in Big Data

For the course:
File Structures (18IS61)

Submitted by:

Amulya L. – 1JS19IS012

Hansika M. – 1JS19IS035

Nidhi Nayak – 1JS19IS053

B. E. Information Science Engineering, Semester: 6th

Under the Guidance of:

Mrs. Sahana V.
Assistant Professor,
Dept. of ISE, JSSATE-B, Bangalore

INDEX

Abstract

Big Data is generally considered as a large collection of data sets (In PetaBytes) which are difficult to analyze, share, store, search, transference through traditional data processing applications.

The rate at which the global data is accumulating and the rapid and continuous interconnecting of people and devices is overwhelming. This has further poses additional challenge to finding even faster techniques of analyzing and mining the big data despite the emergence of specific big data tools.

Indexing and indexing data structures have played an important role in providing faster and improved ways of achieving data processing, mining and retrieval in relational database management systems. The indexing techniques and data structures have the potential of bring the same benefits to big data analytics if properly integrated into the big data analytical platforms.

Chapter 1

Introduction

Data is the key factor today. It includes personal, professional, social data and more. Digitalization and inter-connectivity lead to an unexpected growth of data. The increased use of media and physical networking through sensor networks for business & private purposes generates an enormous amount of data.

This in response changes business processes & open up new opportunities worldwide. The internet is a key driver for data growth. The worldwide generated data already exceed the available storage (Google 2013). Since 2011 interest in an area known as big data has increased exponentially.

Unlike the vast majority of computer science research, big data has received significant public and media interest.

The era of "big data" has opened several doors of opportunities to upgrade science, boost health care services, improve economic growth, reconstruct our educational system, and prepare new types of social interaction and entertainment services. The area of big data is fast-evolving, and is likely to be subject to improvements and amendments in the future.

Conventional data technologies and methods are most of the time slow, expensive and not suitable to handle the storage and the processing of large growing volumes of heterogeneous data. One challenge is to overcome the complex nature of Big Data (volume, velocity and variety).

Nowadays, many open source and proprietary Big Data solutions are available. The goal is to helps decision makers and data scientists to take the next best actions based on discovered patterns, data relations and newly extracted knowledge from Big Data. Traditional searching methods are not adapted to distributed environment and Big Data complexity. Enterprises need to run extensive real-time queries through huge volumes of unstructured and structured data sets.

This demand have led to the development of scalable Search Engines based on appropriate searching and indexing technologies such Lucene and Splunk Processing Language.

Chapter 2

What is Searching and Indexing?

Indexing, as information retrieval technique, is the process of generating all the suitable data structures that allow for efficient retrieval of stored information. While the term index refers to the suitable data structure needed, to allow for the efficient information retrieval [28].

Usually, the data structures used for indexing in most cases do not store the information itself. Rather, it uses other data structures and pointers to locate where the data is actually being stored.

A Meta index is also used to store additional information about stored data like the external name of a document, its length, which can be ranked as the output of the index.

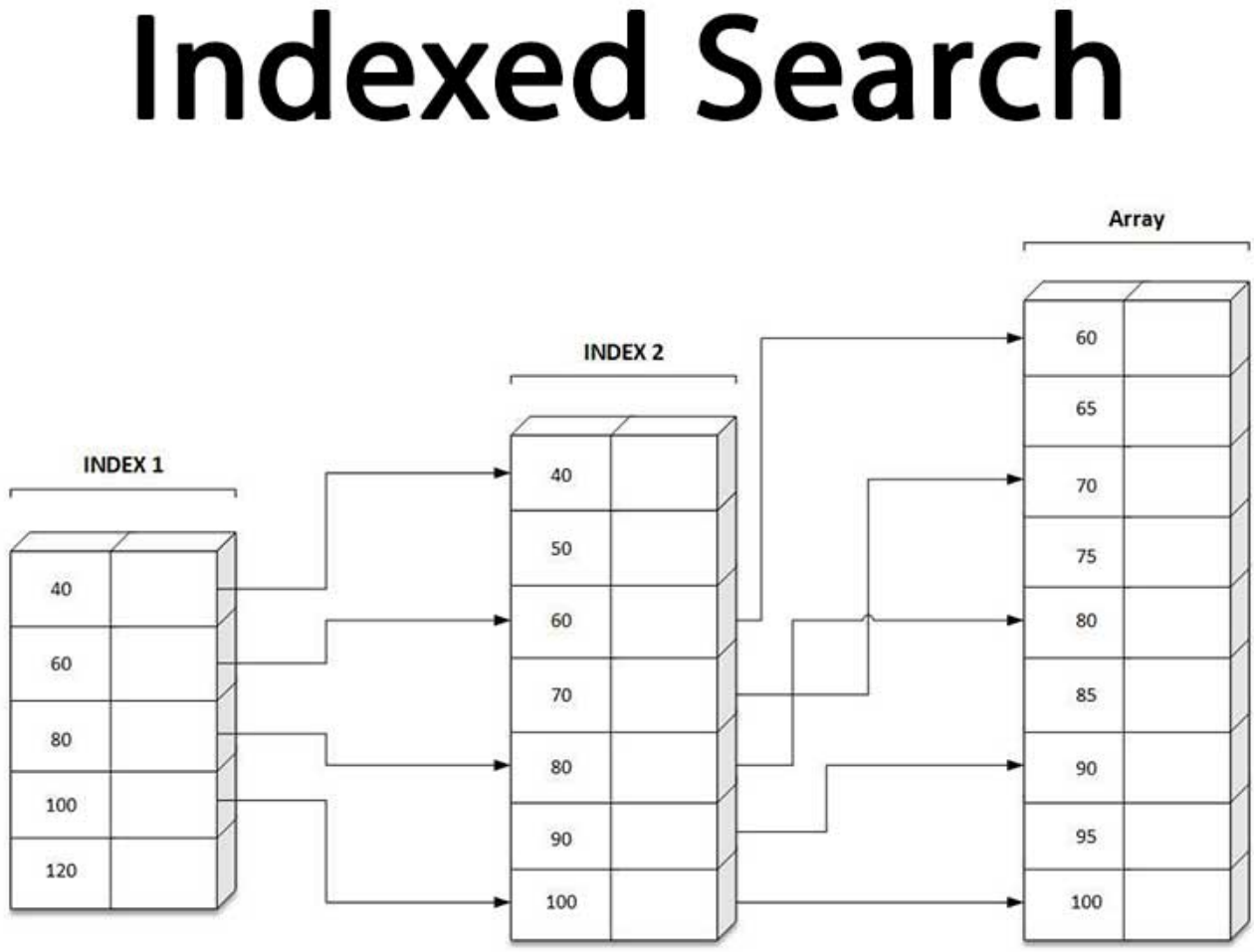

Fig 2.1: Indexed Search

In RDBMSs, the index is the data structure that speeds-up the operation of data retrieval from database tables. The index comes with additional cost for writing/reading it into/from the index table. This leads to more storage usage in order for the extra copy of data created by the index to be maintained.

The indexes are used in locating data quickly by going straight to the data location without having to search all the rows in the database table every time that table is accessed. Indexes are the bases for providing rapid random lookups and efficient access of ordered recorded.

While in Big data, if index is to be implemented the same way as it works with the RDBMS, there is no doubt in will be so big to the extent the the intended fast record retrieval may not be achieved. Therefore, there is need for a closer look at the indexing technique in a way that it will be reasonably small and maintain the targeted goal of faster record retrieval.

One way to think about indices is to consider the following analogy between a search infrastructure and an office filing system. Imagine you hand an intern a stack of thousands of pieces of paper (documents) and tell them to organize these pieces of paper in a filing cabinet (index) to help the company find information more efficiently. The intern will first have to sort through the papers and get a sense of all the information contained within them, then they will have to decide on a system for arranging them in the filing cabinet, then finally they'll need to decide what is the most effective manner for searching through and selecting from the files once they are in the cabinet.

In this example, the process of organizing and filing the papers corresponds to the process of indexing website content, and the method for searching across these organized files and finding those that are most relevant corresponds to the search algorithm.

Chapter 3

Big Data Indexing

Big Data indexes can be said to be a list of tags, names, subjects, etc. of a data-set which references where data can be found. An indexing strategy is the design of an access method to a searched item, or simply put, an index. It also describes how data is organized in a storage system to facilitate information retrieval. The idea of Big Data indexing is to fragment the datasets according to criteria that will be used frequently in query.

The fragments are indexed with each containing value satisfying some query predicates. This is aimed at storing the data in a more organized manner, thereby easing information retrieval. For that we use Dense Indexing.

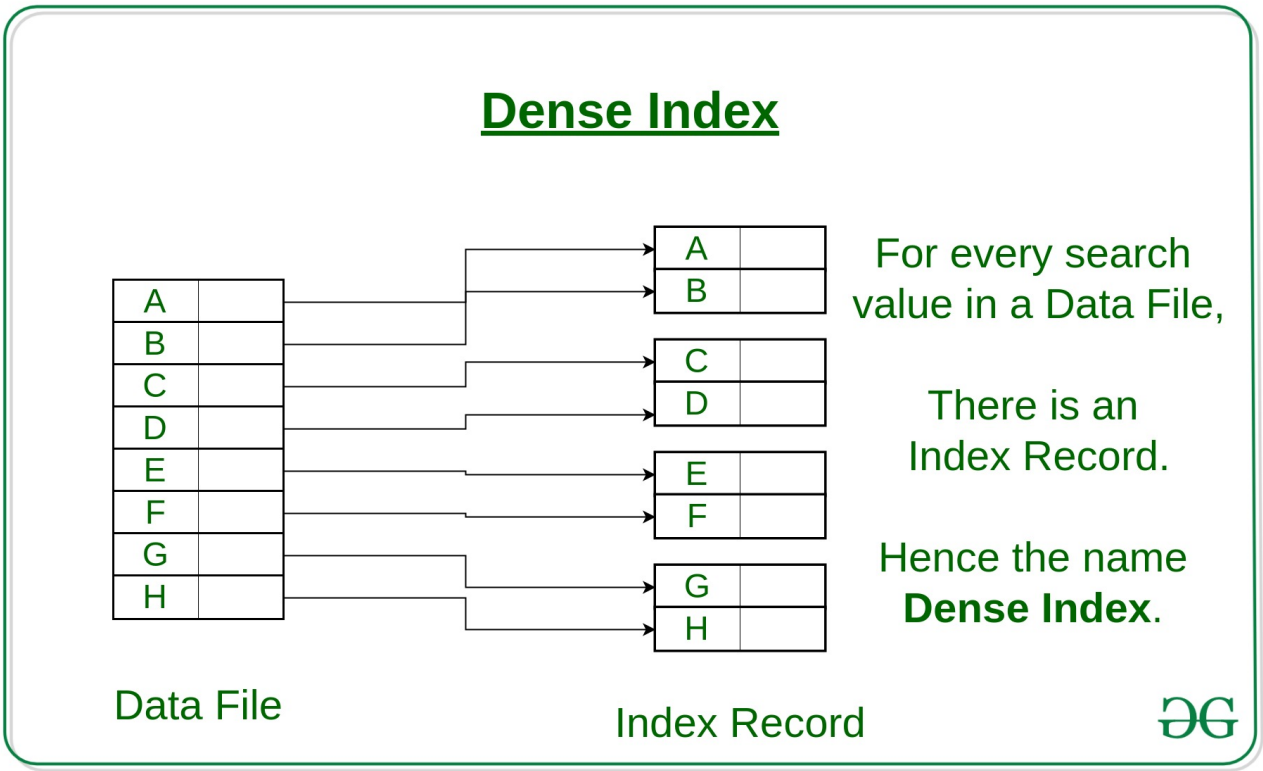

Fig 3.1: Dense Index

Big Data indexing depends on a solution that utilizes a massively parallel computer or machine that interconnects lots of RAM, CPUs, and disk units. The benefits of this are high throughput for data processing, decreased access time for queries, data replication that results in increased availability and reliability, and scalability of the structure.

The design of an access method or the type of indexing strategy to be used in processing a specific data-set depends on the type of queries that will be performed on the data-set, such as similarity queries (nearest neighbor search), range queries, point query, keyword queries, and ad-hoc query. Therefore, the designer must be aware of the type of data to be indexed (e.g. logs, email, audio, video, images, etc.) and the type of query that will be performed on the indexes.

Indexing strategies can be mainly categorized into

- Artificial Intelligence (AI) approach
- Non-Artificial Intelligence (NAI) approach.

There is an additional category Collaborative Artificial Intelligent (CAI) approach which is a hybrid of the above two.

Chapter 4

Artificial Intelligence Approach

Artificial Intelligence (AI) indexing approaches are so called because of their ability to detect unknown behavior in Big Data. They establish relationships between data items by observing patterns and categorizing items or objects with similar traits. Although this gives AI indexing approaches an edge over NAI, the former generally takes more time in information retrieval and are sometimes considered inefficient as compared to NAI indexing approaches.

4.1 Latent Semantic Indexing

Latent Semantic Indexing, LSI for short, is an indexing strategy (retrieval/access method) that identifies patterns between the terms in an unstructured data set (specifically text). It uses a mathematical approach known as Singular Value Decomposition (SVD) for the pattern or relationship
identification. Hence, LSI is not subjected to any language. The main characteristic of LSI is the ability to elicit the conceptual (semantic) content of data sets and to establish relationships between terms with similar contexts. LSI strategy demands very high computational performance as well as memory to index Big Data. LSI supports keyword queries on textual data which can be in the form of web contents (images, audio, etc.), documents, emails, or any item that can be converted into text.

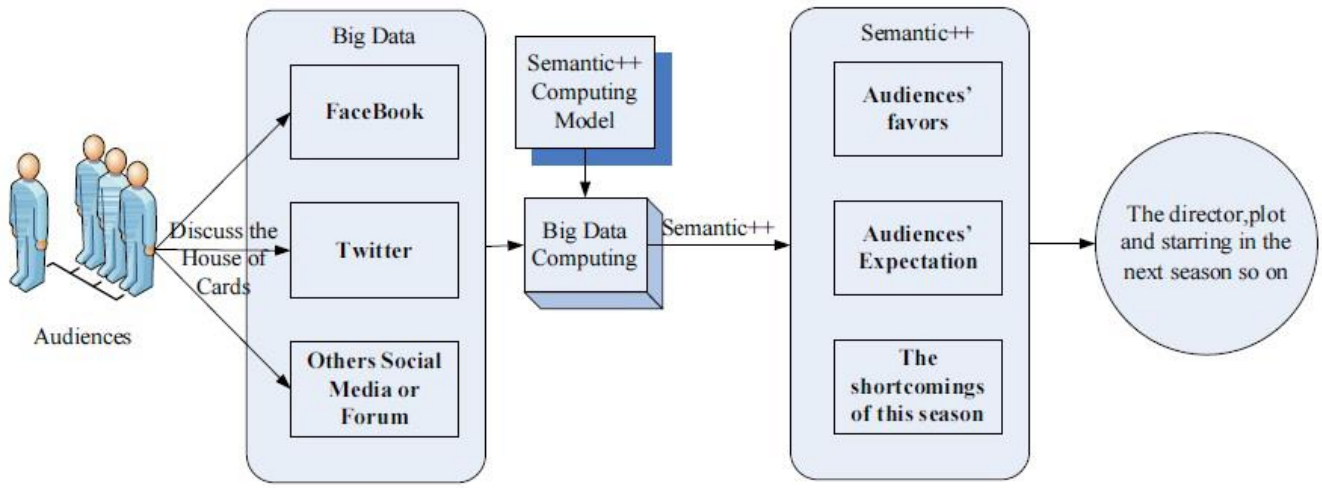

Fig 4.1.1 Latent semantic indexing

4.2 Hidden Markov Model

The Hidden Markov Model (HMM) indexing approach is an access method developed from the Markov model. A Markov model is made up of states which are connected by transitions, where future states are solely dependent on the present state and independent of historical states. Similar to the LSI strategy, the HMM uses pattern recognition and relationship between data. In the HMM indexing approach, data or characteristics which the states depend on

during query, are categorized and stored in advance. The query results are usually predictions of future states of an item, based on the current or present state. The present state is used to predict the future states using the dependent data or characteristics of the states.

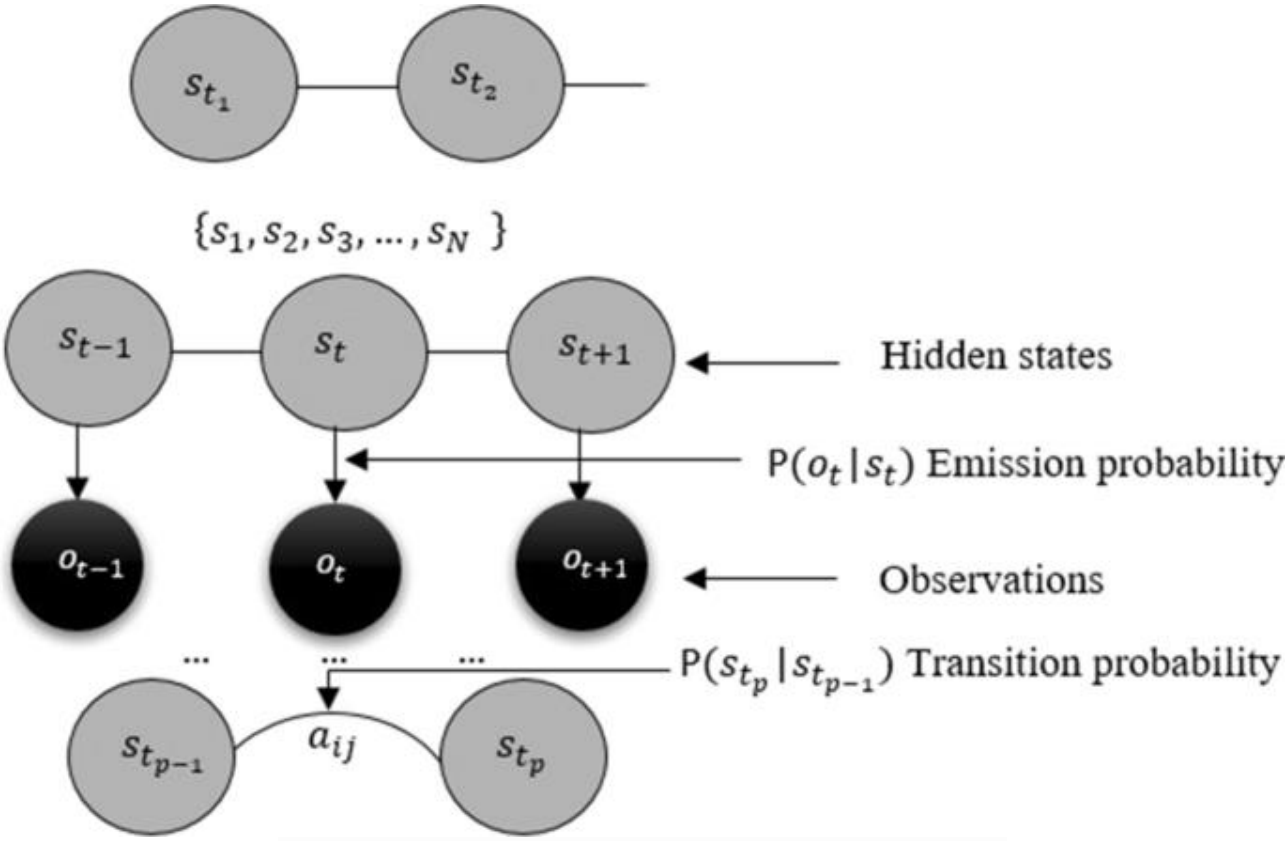

Fig 4.2.1 Graphical representation of a hidden Markov model.

Chapter 5

Non-Artificial Intelligence Approach

In NAI indexing approach, the formation of indexes does not depend on the meaning of the data item or the relationship between texts. Rather, indexes are formed based on items most queried or searched for in a particular data set. The tree-based indexing strategy (B-tree, R-tree and X-tree), inverted indexing approach, hash, and custom (GiST and GIN) indexing, are NAI indexing approaches covered in this paper.

5.1 Tree indexing structures

The Tree indexing structures are the B-tree, R-tree, Xtree, etc. In the Tree indexing strategy, retrieval of data is done in a sorted order, following branch relations of the data item. This satisfies nearest neighbor queries. According to researches, the Tree indexing strategies are being displaced
by other indexing strategies because they are generally outperformed (in terms of speed of information retrieval) by simple sequential scans. The Tree-based indexing strategies are explained as follows:

5.1.1 The B-tree:

A B-tree works like the Binary tree search, but in a more complex manner. This is because the nodes of B-tree have many branches, unlike the binary tree which has two branches per node. So, a B-tree is more complicated than a binary tree. Researches have shown that this strategy is not always fast when searching Big Data and can waste storage spaces since the nodes are not always full.

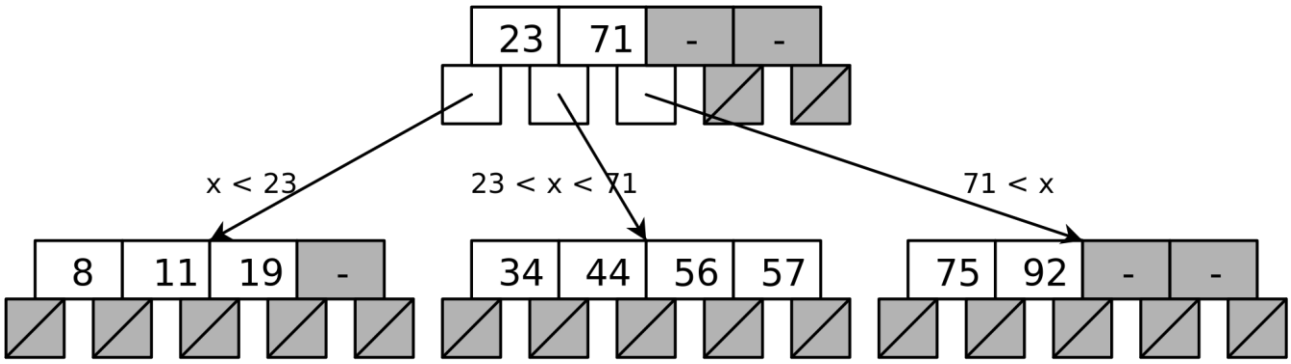

Fig 5.1.1: A basic example of B tree

A B-tree scales linearly, but is only suitable for one dimensional access method unlike other tree-based access m methods or indexing strategies such as the R-tree.
Also, the B-tree algorithm consumes huge computing resources when performing indexing on Big Data. Other variations of the B-tree are the B+tree, B*tree, KDB-tree, and so on.

5.1.2 The R-tree:

This is an indexing strategy used for spatial or range queries. It is mostly applied in Geo-spatial systems with each entry having X and Y coordinates with minimum and maximum values.

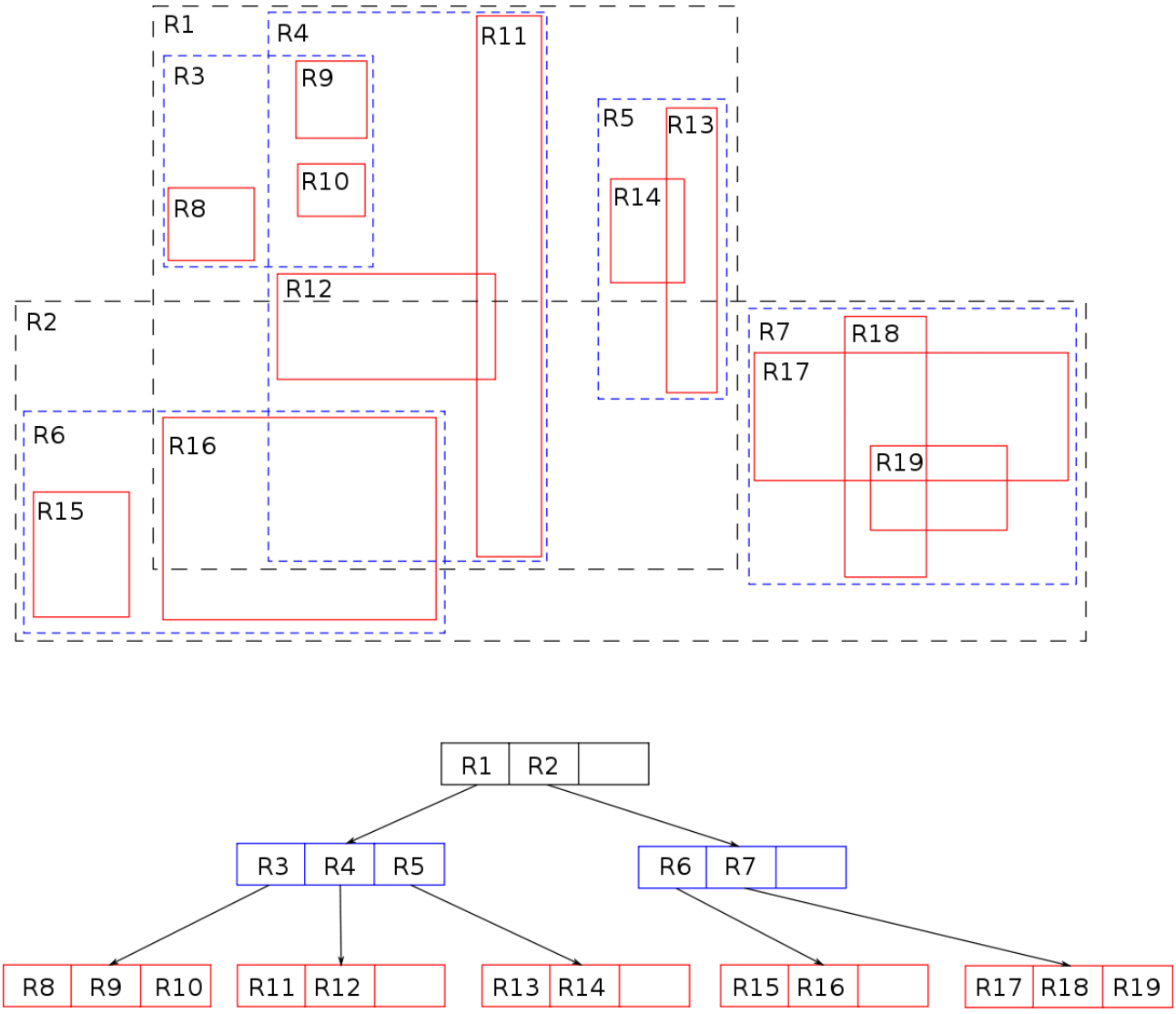

Fig 5.1.2: Example of R-Tree.

The advantage of using an R-tree over a B-tree is that, the R-tree satisfies multi-dimensional or range queries, whereas the B-tree does not. Given a query range, using the R-tree makes finding answers to queries quick.

5.1.3 The X-tree:

This type of indexing strategy, based on the R-tree, satisfies range queries. The X-tree is similar to the R-tree and operates just like the R-tree. Although, unlike the R-tree which satisfies 2-3 dimensional range queries, the X-tree satisfies queries of many dimensions. This implies that the X-tree is a more complicated version of the R-tree. The advantage of the X-tree over the R-tree is that it covers more dimensions, otherwise, the X-tree also consumes memory space due to storage of coordinates.

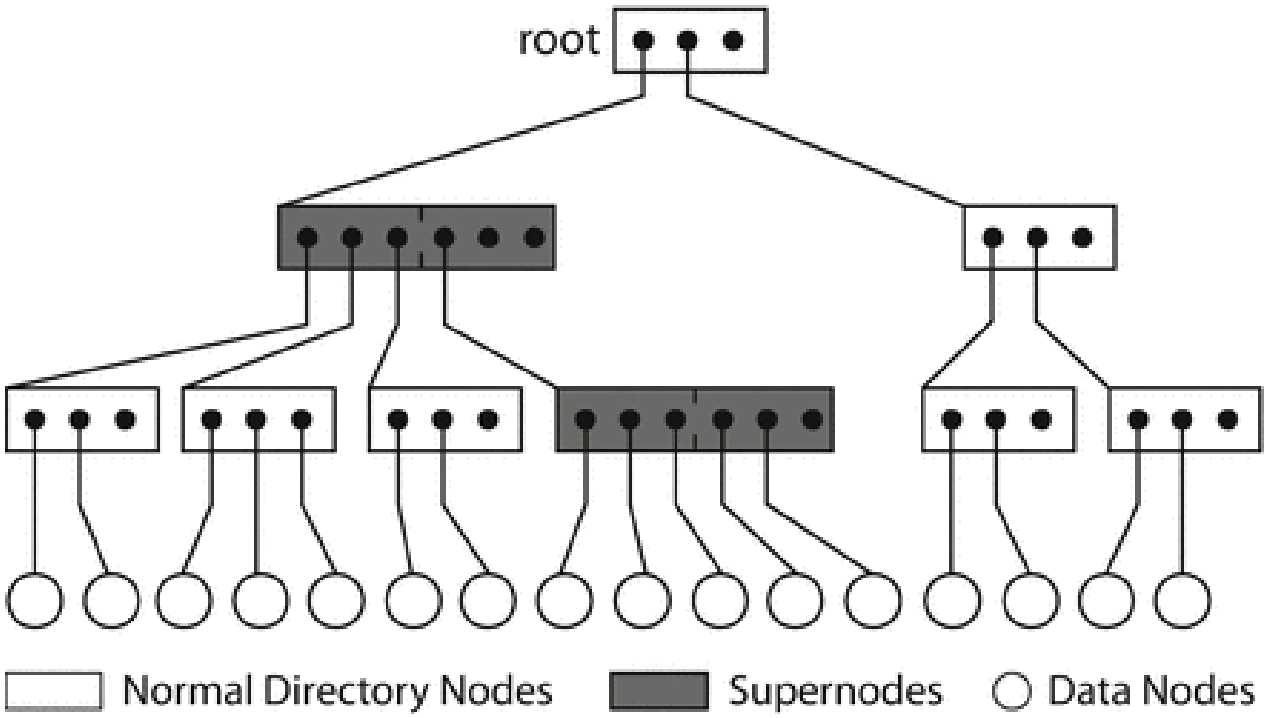

Fig:5.1.3 X-Tree Example.

5.2 Hash Indexing Strategy

Hash allows for equality comparison. Hash indexing accelerates information retrieval by detecting duplicates in a large data-set. Hash is used in Big Data indexing to index and retrieve data items (in a data-set) that are similar to the searched item. It uses a hashed key (which is computed by the hash function and usually shorter than the original value) to store and retrieve indexes. For this reason, hash indexing is more efficient than the tree-based indexing in terms of equality or point query. Simply, search on shorter hashed keys can be faster than search on unpredictable length key (found in tree-based indexes).

Though, hashing technique works fine with limited data size, it tends to exhibit indexing computational overhead as data size increases.

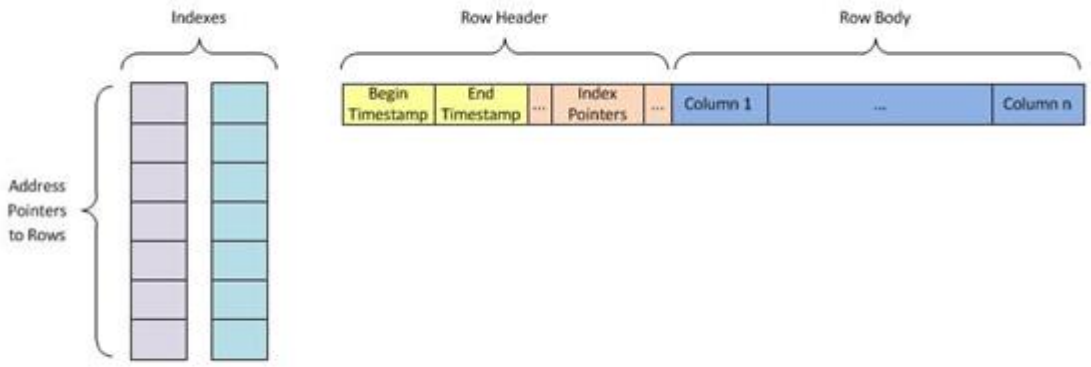

Fig 5.2.1 Hash indexing

5.3 Custom Indexing Strategy

Custom indexing supports multiple field indexing based on arbitrary or user defined indices. They are usually based on indexing strategies such as B-tree, R-tree, inverted index, and hash indexing strategy. Two types of custom indexing strategies are Generalized Search Tree (GiST) and Generalized Inverted Index (GIN).

5.3.1 GiST:

The Generalized Search Tree or GiST indexing strategy, is an indexing strategy based on the B-tree or the R-tree. It allows for the creation of custom or arbitrary fields as indexes. The GiST has the same implementation (for indexing and retrieval) as the R-tree for those based on the Rtree, and as the B-tree for those based on the B-tree. Hence, they support indexing and query on one-dimensional data, as well as multi-dimensional or spatial data.

5.3.2 GIN:

Just as in the GiST, the Generalized Inverted Index or GIN indexing strategy (or access method) uses custom or arbitrary fields as indexes [26]. It is designed for specific user requirements. Though the GIN is implemented like the B-tree and has properties of the inverted index, GIN differs from the B-tree which has comparison-based operations that are predefined. While the B-tree is good for single-match indexes or range queries, the GIN works best for indexes having many duplicates. This is because the GIN queries data only by point or equality matching. This is often viewed as a limitation.

5.4 Inverted Indexing Strategy

Inverted indexing strategy allows for the design of inverted indices which are used for full - text search like what is obtainable in Google and other search engines. An inverted index is made up of a list of all unique words which appear in documents, and a list of documents in which each word appears. With an inverted index, multiple documents can have the same key as index. Also, multiple keys can be used in indexing a document. Inverted indexing is implemented by storing or indexing a set of keypost list pairs, where key is the searched index, and post list is a collection of documents where the key occurs. The problem with inverted indexes is that, two or more words (keys) might be separate terms, but will appear to the user as the same term. Also, synonyms (of the keys) might not be recognized or retrieved during query search.

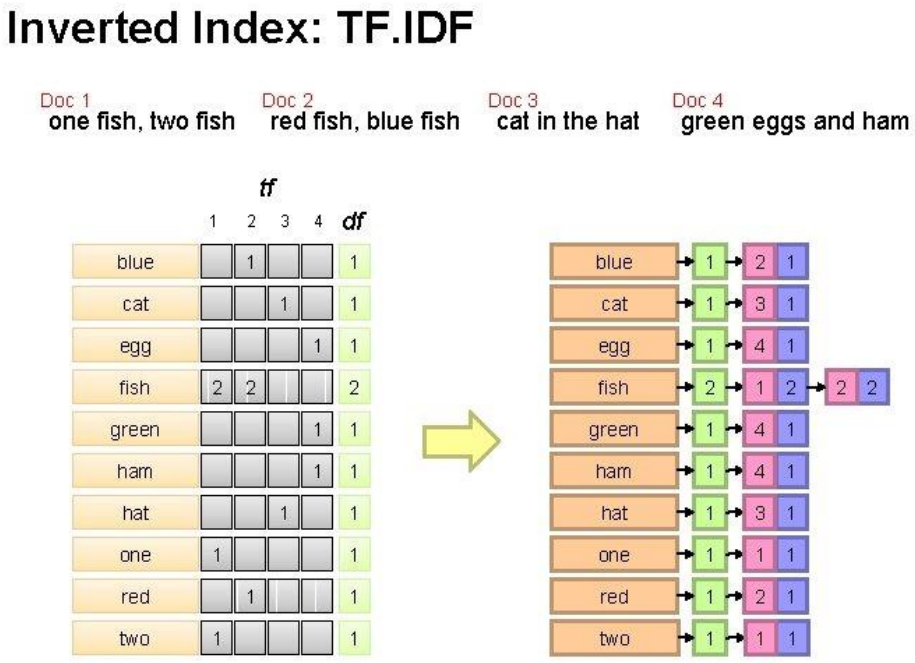

Fig 5.4.1 Basic Example for Inverted Index.

Chapter 6

Collaborative Artificial Intelligent Indexing

CAI based indexing techniques are hybrid of both AI and NAI techniques. Any of the above described NAI and AI method can be combined to efficiently perform indexing in specific domain depending upon the type of data and application feasibility. CAI based indexing out performance AI and NAI techniques while maintaining the trade-off between index construction time and index response accuracy.

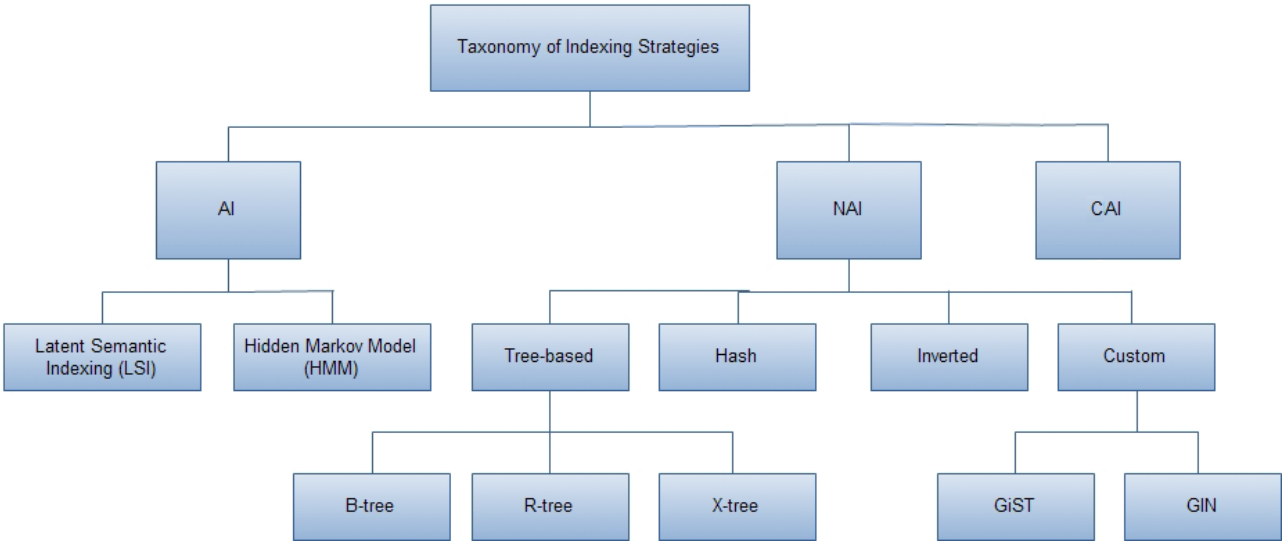

Fig 6.1. Taxonomy of Indexing Strategies

Chapter 7

A look into the world of Parallelism

Besides, indexing strategies, to analyze huge volumes of data, we need to process the data in parallel. Without a robust parallel processing framework, analyzing and crunching Big Data is almost impossible. Let's look at a simple example of why we need parallelism.

A commodity computer can read 50MB/second from hard disk and store 10 terabyte (TB) of data. Let's assume we need to process a job on big data that is 1 TB. A job is basically some processing on a given data set, anything from a set of log files collected from distributed data centers to huge data a web crawler produced for indexing. With a commodity computer which performs sequential read of data from a single drive, the read part itself would take roughly 300 minutes! Given the fact that a computer with large volume of memory and CPU processing power can process the data very fast, I/O operation causes the bottleneck. The obvious solution is parallel reading. Instead of reading the 1 TB data from a single computer, if we read using 500 computers in parallel, we can read at 25GB/seconds.

But parallel processing comes with lots of challenges. Let's try to understand some of the key challenges. If there are 500 computers (nodes) which read data in parallel, what will be the source of data? Will all nodes point to the same source? How do we then split the data such that a specific node reads only specific splits? Do we need to transfer the data to the node over the wire? What is the amount of bandwidth needed to transfer such huge data? Is there any data sharing among nodes?

You can see the challenges of a parallel processing system and the need for a fault tolerant, distributed parallel computing framework which can address all of them. Google made the first breakthrough to solve this massively parallel computing and published a paper on Google File System and Map Reduce in 2004. Doug Cutting got the idea from this paper and started building a similar framework in Java called Hadoop which became a top level Apache project (HTTP://Hadoop.Apache.org/) and open source software. This tool is incorporated to make scalable indexes for Big Data.

Chapter 8

Conclusion

This paper puts together popular data indexing approaches for Big Data processing and management. The objective is to review the potentials of the various indexing strategies and how they are utilized for solving Big data management issues.

Existing techniques for Big Data indexing are classified as Non-Artificial Intelligent, Artificial Intelligent and Collaborative Artificial Intelligent techniques. In general, NAI methods such as B-Tree and variants are suitable for single dimensional data i.e. multimedia data including 1-D images, text, numeric data and log data whereas R-Tree and variants are suitable for Geo spatial data. Bitmap is suitable for low cardinality data.

AI based indexing are suitable for text data including words, sentences and documents. NAI methods perform reasonably. In terms major Big Data characteristics but they are inefficient in detecting the unknown behaviour of big data.
Therefore, AI based intelligent indexing techniques are used to detect and identify the continuously changing behaviour of Big Data but are inefficient as some additional efforts and time are required for training process.

A hybrid approach (combining AI and NAI methods) known as CAI indexing which resolves the above mentioned limitations of NAI and AI techniques while performing efficiently for Big Data. Thus, this report concludes as serving as a guide for studying about the approach best suited in solving a specific problem, and can also serve as a base for the design of more efficient indexing strategies.

REFERENCES:

[1] “REAL TIME SEARCHING OF BIG DATA USING HADOOP, LUCENE, AND SOLR” by Dibyendu Bhattacharya, RSA the Security Division of EMC

[2] “A Survey On Big Data Indexing Strategies” by Fatima Binta Adamu, Adib Habbal, Suhaidi Hassan, R. Les Cottrell, Bebo White, Ibrahim Abdullahi InterNetworks Research Laboratory, Malaysia. SLAC National Accelerator Lab

[3] Desai, Mitali & Mehta, R. & Rana, Dipti. (2019). A Survey on Techniques for Indexing and Hashing in Big Data. 10.1109/CCAA.2018.8777454.

[4] “Analysis, Searching & Sorting in Big data using Apache Hadoop” IJSTE - International Journal of Science Technology & Engineering | Volume 2 | Issue 09 | March 2016

[5] Ali, Usman & Ahmad, Rohiza & Zakaria, Nordin. (2021). Indexing in Big Data Mining and Analytics. 10.1007/978-3-030-66288-2_5.