



**TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
PULCHOWK CAMPUS**

THESIS NO: PUL075MSCSK020

Optimization of Shard Selection Techniques on Elasticsearch

**by
Yashasvi Raj Pant**

**A THESIS
SUBMITTED TO THE DEPARTMENT OF ELECTRONICS AND
COMPUTER ENGINEERING IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN
COMPUTER SYSTEM AND KNOWLEDGE ENGINEERING**

**DEPARTMENT OF ELECTRONICS AND COMPUTER ENGINEERING
LALITPUR, NEPAL**

August, 2021

Optimization of Shard Selection Techniques on Elasticsearch

by

Yashasvi Raj Pant

075MSCSK020

Thesis Supervisor

Assoc. Prof. Dr. Nanda Bikram Adhikari

A thesis submitted in partial fulfillment of the requirements for the
degree of Masters of Science in Computer System and Knowledge
Engineering

Department of Electronics and Computer Engineering

Institute of Engineering, Pulchowk Campus

Tribhuvan University

Lalitpur, Nepal

August, 2021

COPYRIGHT©

The author has agreed that the library, Department of Electronics and Computer Engineering, Institute of Engineering, Pulchowk Campus, may make this thesis freely available for inspection. Moreover the author has agreed that the permission for extensive copying of this thesis work for scholarly purpose may be granted by the professor(s), who supervised the thesis work recorded herein or, in their absence, by the Head of the Department, wherein this thesis was done. It is understood that the recognition will be given to the author of this thesis and to the Department of Electronics and Computer Engineering, Pulchowk Campus in any use of the material of this thesis. Copying of publication or other use of this thesis for financial gain without approval of the Department of Electronics and Computer Engineering, Institute of Engineering, Pulchowk Campus and author's written permission is prohibited.

Request for permission to copy or to make any use of the material in this thesis in whole or part should be addressed to:

Head

Department of Electronics and Computer Engineering

Institute of Engineering, Pulchowk Campus

Pulchowk, Lalitpur, Nepal

DECLARATION

I declare that the work hereby submitted for Master of Science in Computer System and Knowledge Engineering (MSCSKE) at IOE, Pulchowk Campus entitled “**Optimization of Shard Selection Techniques on Elasticsearch**” is my own work and has not been previously submitted by me at any university for any academic award.

I authorize IOE, Pulchowk Campus to lend this thesis to other institution or individuals for the purpose of scholarly research.

Yashasvi Raj Pant

075MSCSK020

Date: August, 2021

RECOMMENDATION

The undersigned certify that they have read and recommended to the Department of Electronics and Computer Engineering for acceptance, a thesis entitled “**Optimization of Shard Selection Techniques on Elasticsearch**”, submitted by **Yashasvi Raj Pant** in partial fulfillment of the requirement for the award of the degree of “**Master of Science in Computer System and Knowledge Engineering**”.

.....

Supervisor:

Assoc. Prof. Dr. Nanda Bikram Adhikari

Department of Electronics and Computer Engineering,

Institute of Engineering, Tribhuvan University

.....

External Examiner:

Deepen Chapagain

CISSP | Country Director at Logpoint, Nepal

.....

Committee Chairperson:

Assoc. Prof. Dr. Nanda Bikram Adhikari

Department of Electronics and Computer Engineering,

Institute of Engineering, Tribhuvan University

Date: August, 2021

DEPARTMENTAL ACCEPTANCE

The thesis entitled “**Optimization of Shard Selection Techniques on Elasticsearch**”, submitted by **Yashasvi Raj Pant** in partial fulfillment of the requirement for the award of the degree of “**Master of Science in Computer System and Knowledge Engineering**” has been accepted as a bonafide record of work independently carried out by him in the department.

.....

Prof. Dr. Ram Krishna Maharjan

Head of the Department

Department of Electronics and Computer Engineering

Pulchowk Campus

Institute of Engineering,

Tribhuvan University,

Nepal.

ACKNOWLEDGEMENT

Firstly, I owe my sincere gratitude to the Department of Electronics and Computer Engineering, IOE, Pulchowk Campus for providing me with an opportunity to work on the thesis as part of our syllabus for Masters of Science in Computer System and Knowledge Engineering(MSCSKE).

I am very grateful to my supervisor as well as program coordinator **Assoc. Prof. Dr. Nanda Bikram Adhikari** for his invaluable guidance, timely responses, and suggestions throughout this thesis.

I would also like to express my sincere gratitude to my external examiner **Deepen Chapagain** for his guidance and suggestions for the further improvement of this thesis.

Also, a very special and heartily thankful to the **Asst. Prof. Dr. Basanta Joshi** for providing valuable guidance and suggestions for this thesis.

ABSTRACT

Distributed systems typically consist of several nodes connected together for handling search operations. Data is divided into those nodes for the purpose of parallel processing and replications. Elasticsearch is the popular distributed search engine where data is organized into indices. Each index of Elasticsearch consists of one or more shards and those shards can be distributed over different nodes. When a search operation is performed on a particular index, sending the search requests to all the related shards distributed over different nodes might result in high latency especially when the size of the cluster is large and nodes are far apart. Shard Selection is the technique that attempts to forward the query to the highly relevant shards discarding other non-relevant shards and thus decreasing the latency. Shard selection comes with the cost of relevance, it's obvious that the application of the shard selection algorithm might decrease the query relevance. There are several shard selection algorithms developed time and again. Among them, ReDDe, Sushi, and Rank-S are very popular. In this paper, a new shard selection algorithm called Hybrid Optimized Shard Selection Algorithm (HOSSA) is developed extracting core features from each of these three algorithms and also optimizing shard-related parameters. HOSSA has shown improvements both in terms of latency and relevance compared to the existing shard selection algorithms.

The experimentation is performed using Insider Threat Test Dataset(CERT V6.2) collected from Carnegie Mellon University site . In terms of average latency, the HOSSA is performing 19.34%, 15.6%, and 7.30% better than SUSHI, ReDDe, and Rank-S respectively. In terms of Average Document Score, the HOSSA is performing 33.09%, 18.89%, and 3.31% better than SUSHI, ReDDe, and Rank-S respectively.

Keywords: Nodes, Elasticsearch, Index, Shards, ReDDe, Sushi, Rank-S

TABLE OF CONTENTS

COPYRIGHT	iii
DECLARATION	iv
RECOMMENDATION	v
DEPARTMENTAL ACCEPTANCE	vi
ACKNOWLEDGEMENT	vii
ABSTRACT	viii
TABLE OF CONTENTS	ix
LIST OF FIGURES	xi
LIST OF TABLES	xiv
LIST OF ABBREVIATIONS	xv
1 INTRODUCTION	1
1.1 Background and Motivation	1
1.2 Problem Definition	3
1.3 Objectives	4
1.4 Scope of the Work	4
1.5 Originality of this Work	4
1.6 Organisation of thesis work	4
2 LITERATURE REVIEW	6
3 METHODOLOGY	10
3.1 Data Collection	11
3.2 Data Processing	12
3.3 Parameters	12
3.4 Indexing and Searching	13

3.5	Hybrid Optimized Shard Selection Algorithm	14
3.5.1	Architecture Design	14
3.5.2	Algorithm Implementation	15
3.5.3	Tunable shard selection Elasticsearch plugin	18
3.5.4	Special Cases	19
3.6	Evaluation Metrics	19
3.6.1	Document Score	19
3.6.2	Search Latency	20
3.6.3	Precision at K (P@K)	21
3.6.4	Recall at K (P@K)	21
3.7	Validations	21
3.8	Experimental Setup	22
3.9	Tools and Resources	22
4	RESULTS AND DISCUSSION	24
4.1	Experimental Results and Discussions	24
4.2	Validation Results	37
5	CONCLUSION AND FUTURE ENHANCEMENT	38
5.1	Conclusion	38
5.2	Future Works	40
	REFERENCES	42
	APPENDIX A	43
	APPENDIX B	47

LIST OF FIGURES

1.1	Figure showing the distribution of primary and replica shards of an index over different nodes	2
1.2	A schematic show of Shard Selection Problem	3
3.1	Implementation Flow Chart for the development of Hybrid Optimized Shard Selection Algorithm	10
3.2	A schematic representation of data processing steps to Elasticsearch via FileBeat and Logstash	12
3.3	System Architecture for the implementation of HOSSA	15
3.4	Block Diagram showing formation of HOSSA	15
3.5	Diagram illustrating the core feature of ReDDe, Sushi, and Rank-S extracted for the formation of HOSSA	18
4.1	Search Latency (ms) against Maximum number of shards allowed to select for different shard selection algorithms on 1st query	24
4.2	Search Latency (ms) against Maximum number of shards allowed to select for different shard selection algorithms on 2nd query . . .	25
4.3	Search Latency (ms) against Maximum number of shards allowed to select for different shard selection algorithms on 3rd query . . .	25
4.4	Average Document Score against Maximum number of shards allowed to select for different shard selection algorithms on 1st Query	26
4.5	Average Document Score against Maximum number of shards allowed to select for different shard selection algorithms for 2nd Query	26
4.6	Average Document Score against Maximum number of shards allowed to select for different shard selection algorithms for 3rd Query	27

4.7	Precision@10 against Maximum number of shards allowed to select for different shard selection algorithms for 1st Query	27
4.8	Precision@30 against Maximum number of shards allowed to select for different shard selection algorithms for 1st Query	28
4.9	Precision@10 against Maximum number of shards allowed to select for different shard selection algorithms for 2nd Query	29
4.10	Precision@30 against Maximum number of shards allowed to select for different shard selection algorithms for 2nd Query	29
4.11	Precision@10 against Maximum number of shards allowed to select for different shard selection algorithms for 3rd Query	30
4.12	Precision@30 against Maximum number of shards allowed to select for different shard selection algorithms for 3rd Query	30
4.13	Recall@10 against Maximum number of shards allowed to select for different shard selection algorithms for 1st Query	31
4.14	Recall@30 against Maximum number of shards allowed to select for different shard selection algorithms for 1st Query	31
4.15	Recall@10 against Maximum number of shards allowed to select for different shard selection algorithms for 2nd Query	32
4.16	Recall@30 against Maximum number of shards allowed to select for different shard selection algorithms for 2nd Query	32
4.17	Recall@10 against Maximum number of shards allowed to select for different shard selection algorithms for 3rd Query	33
4.18	Recall@30 against Maximum number of shards allowed to select for different shard selection algorithms for 3rd Query	33
4.19	Search Latency (ms) against α , a tunable constant considered for voting	35
4.20	Average Document Score against α , a tunable constant considered for voting	36

4.21 Search Latency(ms) against Heap Memory(GB)	36
---	----

LIST OF TABLES

3.1	Table showing descriptions of datasets for algorithm development and parameter optimization	11
3.2	Table showing descriptions of datasets for validations	11
3.3	Table showing Elasticsearch Parameters for optimization	13
3.4	AWS Elasticsearch Server specification for the experimentation . . .	22
3.5	AWS Elasticsearch Server specification for the validation	22
4.1	Table showing average value of the experimentation results obtained from 15 queries	34
4.2	Table showing average value of the validation results obtained from three queries	37

LIST OF ABBREVIATIONS

AWS	Amazon Web Services
CCI	Centralized Complete Index
CORI	Collection Retrieval Inference Network
CRUD	Create, Read, Update and Delete
CSI	Centralized Sample Index
HOSSA	Hybrid Optimized Shard Selection Algorithm
IDE	Integrated Development Environment
ReDDE	Relevant Document Distribution Estimation Method for Resource Selection
REST	Representational state transfer
SHiRE	Sampling-Based Hierarchical Relevance Estimation
SUSHI	Scoring Scaled Samples for Server Selection

CHAPTER 1

INTRODUCTION

1.1 Background and Motivation

Parallel processing is the core of any distributed system. To perform parallel processings, a distributed system consists of several nodes linked across a network. To increase the performance of the distributed system, data is divided into chunks across those nodes which are called shards. Hence, sharding is the technique to divide the data into chunks creating partitions of the data and distributing it across the nodes for the purpose of parallelism and replications [1].

Elasticsearch is one of the distributed, Lucene-based, scalable, open-source search engines where sharding is used. It is capable of performing RESTful CRUD operations in a distributed manner for a huge amount of data.

Shard is the core of the distributed system of Elasticsearch. There are two types of shards in Elasticsearch: primary shard(commonly referred to as Shard) and replica shard(commonly referred to as Replica). Each shard is a single Lucene Index of Elasticsearch.

Each document of the Elasticsearch belongs to a primary shard whereas a replica shard is the copy of the primary shard. Elasticsearch is actually a grouping of one or more shards and the shard is actually a self-contained index.

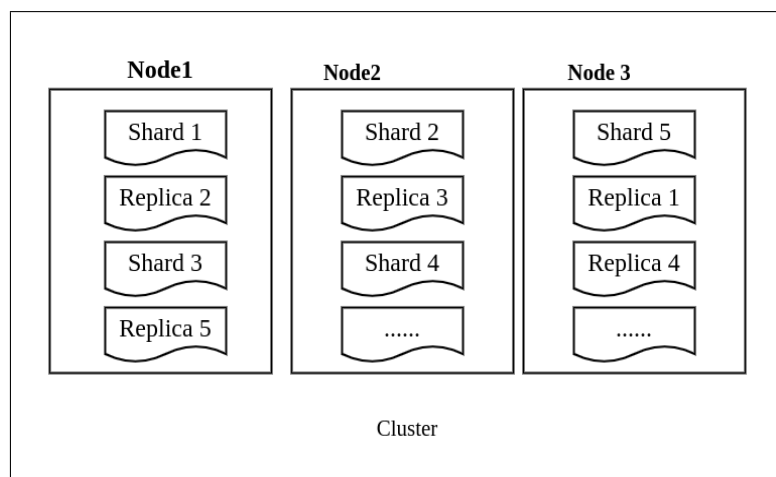


Figure 1.1: Figure showing the distribution of primary and replica shards of an index over different nodes

Shard selection is one of the optimization techniques for distributed search engines like Elasticsearch. The key concept of the Shard selection technique is to process the user query only to the shard that contains relevant user documents while ignoring others. Shard selection typically consists of the concept of broker nodes and data nodes. A data node stores data, process search queries and return the relevant document. Each data node contains one or more shards. A broker node is responsible for receiving the search query from the user, re-routing the query to the appropriate data nodes, receive the result from those nodes, combine and return it to the user. The concept of Shard Selection is shown in the figure 1.2.

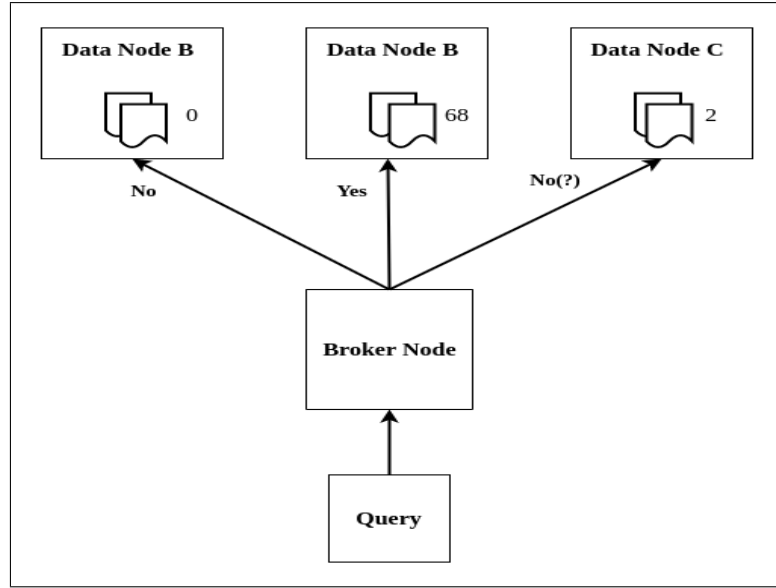


Figure 1.2: A schematic show of Shard Selection Problem

In the above figure, a proper shard selection will discard Node A and Node C, since shards in both of the nodes contain little or no relevant document and route the query only toward Node B.

1.2 Problem Definition

Elasticsearch typically consists of a large number of nodes and shards. When a search query is executed on an index, the query is executed in all the shards of that index, which might be distributed across different nodes, by default. Sending the query to all the shards across different nodes might require high latency for the large size cluster when the nodes are far apart. To forward queries to the highly relevant shards, shard selection methods are used. When using the shard selection technique, the search relevance might be negatively impacted because some shard results might be lost due to the shard selection. Also, the shard selection process might be based on user requirements. For example, User A might need fast search results compromising some relevance in the data while User B might need high relevant data even though search time is slow. Some previous research is done to implement Shard Selection techniques to decrease the latency and increase the relevance. But those researches are done on constant initial Elasticsearch parameters and on a specific user preference. So, this thesis will be an iterative

process to combine and improve different Shard Selection algorithms based on user preferences along with the optimization of shard-related Elasticsearch parameters.

1.3 Objectives

The objectives of this thesis are:

- To study, optimize and combine different shard selection algorithms along with the respective shard-related parameters for Elasticsearch Optimization in terms of query performance and relevance.
- To develop the tunable shard selection Elasticsearch plugin which can be configured according to the user preferences.

1.4 Scope of the Work

Performance Distributed system is the major concern in today's world. The thesis is an attempt to work in a portion of the distribution system called Shard Selection. The scope of the project covers the development of a new shard selection algorithm using existing Shard Selection Algorithms along with the optimization of shard-related parameters. A popular distributed search engine called Elasticsearch is chosen for this purpose.

1.5 Originality of this Work

The thesis introduced a new algorithm called HOSSA using the existing popular algorithms along with parameter optimization. Also, a tunable shard selection Elasticsearch plugin is developed which can be configured according to the user preferences.

1.6 Organisation of thesis work

The thesis is organized into five chapters. In chapter one, a brief introduction and motivation of the problem are given along with the objective of doing research in

this particular field. Chapter two gives the past related works in a similar domain. Chapter three describes the detailed process in performing the research along with the detail of the data used. Chapter four shows the experimental results along with related discussions. Chapter five draws conclusions of the thesis, along with the limitations and future works that can be done.

CHAPTER 2

LITERATURE REVIEW

Scaling and Optimization of Distributed Systems and Big Data Technology [2, 3, 4] has been the major topic of interest since the early 90s. Among those, Shard related optimization is one of the major interests in the distributed system. Hence, many researches are being performed in a distributed search engine for the shard selection. For instance, the query-routing approach [5] is one of them. This approach explains how a query is routed to nodes for the relevant retrieval of the results.

There are many shard selection algorithms proposed till now. Collection Retrieval Inference Network (CORI) [6] was considered to be the first successful shard selection algorithm among them introduced in the mid-90s. It is based on Lexicon algorithms and based on a probabilistic model for information retrieval called a Bayesian network. Bayesian networks have been refined a lot since then.

HighSim [7] is considered the best algorithm among a range of lexicon shard selection algorithms. It is an optimistic assumption-based algorithm. It assumes that a single document of the shard might contain all the terms from a query.

Other different types of algorithm than Lexicon Algorithms are the Surrogate algorithms [7, 8]. It was developed originally to work in distributed uncooperative search engines.

Best-N algorithm [8] is one of the Surrogate algorithms where the goodness for each term is calculated for each document. In this algorithm, the first N terms ranked in terms of goodness is considered.

Another surrogate algorithm developed for an uncooperative environment is the Rel-

evant Document Distribution Estimation Method for Resource Selection (ReDDE) [9]. This algorithm is considered as benchmark for the shard selections. In this algorithm, a centralized index called CCI at the broker node is formed which contains sampled documents from all the shards in the data node. When a query from the user arrives at the broker node, estimation is done on the number of documents relevant to the query of each shard, and shards are ranked accordingly. The number of documents relevance is given as:

$$Rel(S_{i,q}) = \sum_{d \in S_{i_sampl}} P(Rel|d) \times \frac{|S_i|}{|S_{i_sampl}|} \quad (2.1)$$

$Rel(S_{i,q})$ = Number of document relevant to query q in shard S_i

q = query in a shards over the documents-collection S_i

$P(Rel|d)$ = The estimated probability of relevance for document d to query q in CCI

$P(d|S_i)$ = Prior probability of document d in shard S_i

S_{i_sampl} = set of sample document from shard S_i

To central rank of the document is approximated using the formula:

$$rank_central(d_i) = \sum_{rank_samp(d_j) < rank_samp(d_i)} \frac{|S_j|}{|S_{j_sampl}|} \quad (2.2)$$

$P(Rel|d)$ is calculated as:

$$P(Rel|d) = \begin{cases} \gamma, & \text{if } rank_central(d) < \beta \times |S_i|. \\ 0, & \text{otherwise.} \end{cases} \quad (2.3)$$

And, the estimated relevance of given shard S_i on query q can be found with the goodness-score given as:

$$goodness(S_{i,q}) = \frac{Rel(S_i, q)}{\sum_j R(S_{j,q})} \quad (2.4)$$

Sushi [10] is one of the latest surrogate algorithms. Sushi rank shards according to the centralized sample index based on two steps Rank adjustment and curve-fitting. The Rank Adjustment formula for Sushi is given as:

$$rank_adjusted(d) = (rank_sample(d) + 0.5) \times \frac{|c|}{|S_c|} \quad (2.5)$$

Which shows the ratio of the size of the shard they belong to i.e. $|c|$ to the size of the sample from that shard $|S_c|$.

SHiRE [11] is another surrogate algorithm concept that utilizes CSI. In these algorithms, a bottom-up traversing hierarchy is followed for shard ranking. Voting to the shard is performed when a document is fetched from the shard.

$$Vote(d) = S \times B^{-U} \quad (2.6)$$

S = score of the document given from the CSI ranking

B = exponential base

U = level at which the document was found in the hierarchy

In the Lexicon SHiRE (lex-s), the concept of lexical similarity between sample documents is used. It uses similarity to construct the hierarchy. On the other hand, in the Connected SHiRE (conn-s), shard-membership of the documents is used to construct the hierarchy.

Ranked SHiRE (rank-s) is considered the simplest hierarchy among the SHiRE family.

Some of the earlier research was done focusing on the performance and optimization of the above algorithms. In the paper [12], comparative performance analysis is done using Redde, Rank-s, and CORI shard selection algorithms. In the paper [13], a shard selection plugin called SAFE was developed that takes and analyzes

ReDDe, HighSim, Sushi, and Rank-S to improve the performance. In the paper [14], resource selection, score normalization, and result merging techniques for small documents are analyzed. In the paper [15], a quantitative model of shard, placement strategy of the shard, and local balancing were considered for the optimization. Similarly, in the paper [16], a similar approach for Auto-Sharding is done on another NoSQL technology MongoDB.

CHAPTER 3

METHODOLOGY

The thesis is an approach to improve the existing shard selection algorithm and optimize the shard-related parameters. The work involves the combination and optimization of existing three popular shard selection algorithms namely called ReDDE, Sushi, and Rank-S to produce an improved hybrid algorithm here referred to as HOSSA.

It will be an iterative approach based on different shard-related parameters and user preferences. Combining and optimizing different shard selection algorithms along with the shard-related parameters is the main motive here.

The implementation flow chart is shown in the figure 3.1 :

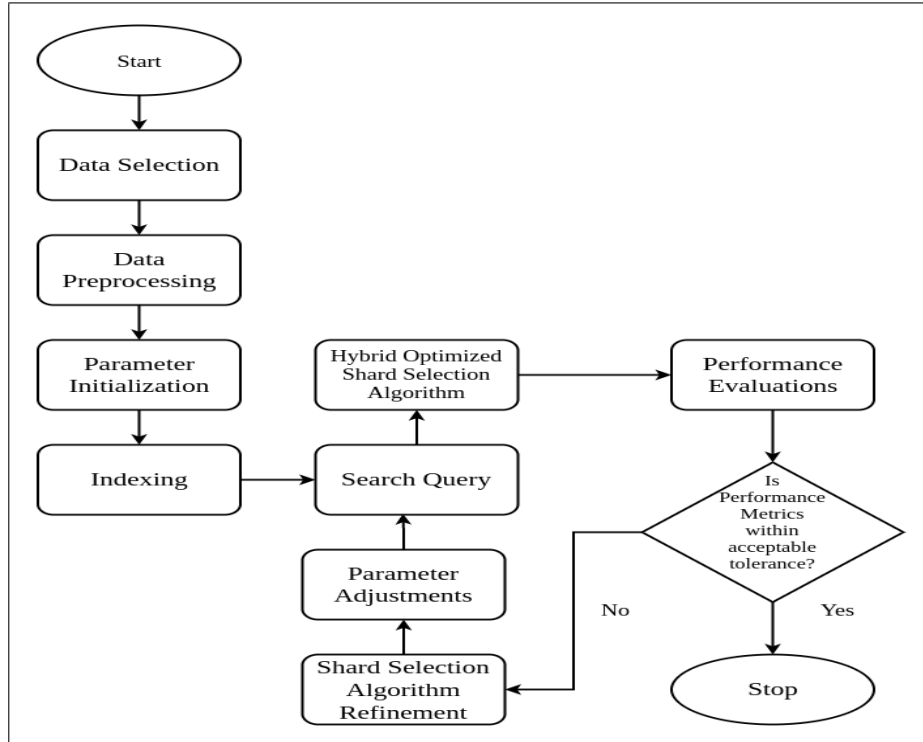


Figure 3.1: Implementation Flow Chart for the development of Hybrid Optimized Shard Selection Algorithm

The above process is summarized in the below steps:

3.1 Data Collection

For the algorithm development and parameter optimizations, data collection is done from Insider Threat Test Dataset(CERT V6.2) from Carnegie Mellon University site which contains up to 100 GB of data of Insider Threats. It is suitable for our experimentation purpose as it has a large variety of data sets suitable for the full-text search needed for the experimentation.

Following is the description of the dataset used for algorithm development and parameter optimizations.

Table 3.1: Table showing descriptions of datasets for algorithm development and parameter optimization

File Name	Total Row Count	Total Data Size	Indexed Document Count
email.csv	10994958	8.1 GB	150000
http.csv	117025217	90.2 GB	100000

Similarly, Shakespeare’s play dataset used in the paper [12] is collected from the kaggle for validation purposes.

Following is the description of the dataset used for the validation.

Table 3.2: Table showing descriptions of datasets for validations

File Name	Total Row Count	Total Data Size	Indexed Document Count
Shakespeare_data.csv	111397	10.2 MB	10000

3.2 Data Processing

CSV format raw data collected from the above source are fed to Logstash, a pipeline to process the data. Large size data is fed to Logstash via a lightweight shipper called Filebeat which is another server that saves the memory of the main server from which data is fed while small data is fed directly to logstash. In logstash, some initial processings(Parsing, Filtering, and Transforming) are done to the raw data to make it Elasticsearch compatible. Then, the refined data is indexed to Elasticsearch via Logstash. The data processing is shown in the block diagram:

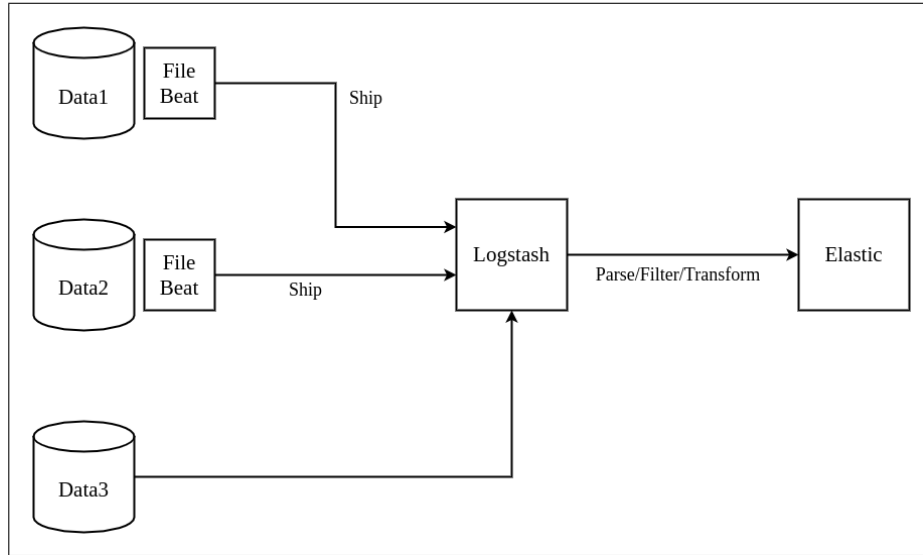


Figure 3.2: A schematic representation of data processing steps to Elasticsearch via FileBeat and Logstash

3.3 Parameters

Following are the parameters that are used. They are initialized with the default values and updated during the process to get optimum parameters.

Table 3.3: Table showing Elasticsearch Parameters for optimization

Parameter	Description
index.number_of_replicas	number of replicas of each primary shard
indices.memory.index_buffer_size	allocation of heap memory (percentage or byte size)
index.number_of_shards	number of primary shards of an index
alpha(α)	a tunable numeric constant to determine how many documents are considered for the voting of their priority

3.4 Indexing and Searching

Data is indexed to Elasticsearch via logstash as stated in the Data Pre-Processing step. The fields that need to be queried using the shard selection algorithm need to be specified in the configuration of the node. The algorithm only supports string fields. A precomputed CSI is processed using those fields during the sync operation between data nodes and broker nodes. Standard Analyzer is used for the experimentation process.

While searching using those specified fields, the CSI is again computed against the query to estimate and route to the relevant shards.

For searching, full text searches are performed using Match Query on indices *http* and *email* to get the relevant documents. Total of 15 queries are executed for the experimentation. Following are the description of 3 of queries executed. Other queries are included in APPENDIX A:

Query1:

Query Type: Match Query

Query Index: http

Field Queried: content

Search String: “players coaches”

Query2:

Query Type: Match Query

Query Index: email

Field Queried: message

Search String: “greeting from here”

Query3:

Query Type: Match Query

Query Index: email

Field Queried: message

Search String: “hello world”

3.5 Hybrid Optimized Shard Selection Algorithm

3.5.1 Architecture Design

Elasticsearch has a concept of coordinating node, whose primary function is to act as a smart load balancer and forwards the request to the data nodes which hold the data. Since the broker node in shard selection has a similar function, coordinating nodes are treated as the broker node. Iterative experimentation is performed with the performance testing using Jmeter to get the optimum number of shards and nodes.

For the experimentation purpose, an Elasticsearch cluster consisting of ten data nodes and two broker nodes is designed. Two broker nodes are added considering the fault tolerance and redundancy aspects. In case, a broker node fails, another broker node can handle the shard selection task. In the almost unlikely case, if both the broker node fails, the search operation takes place via default search. As mentioned above, two indices namely email and http are set such that each node contains a primary shard and a replica shard of both indices.

The architecture design is shown in the figure 3.3.

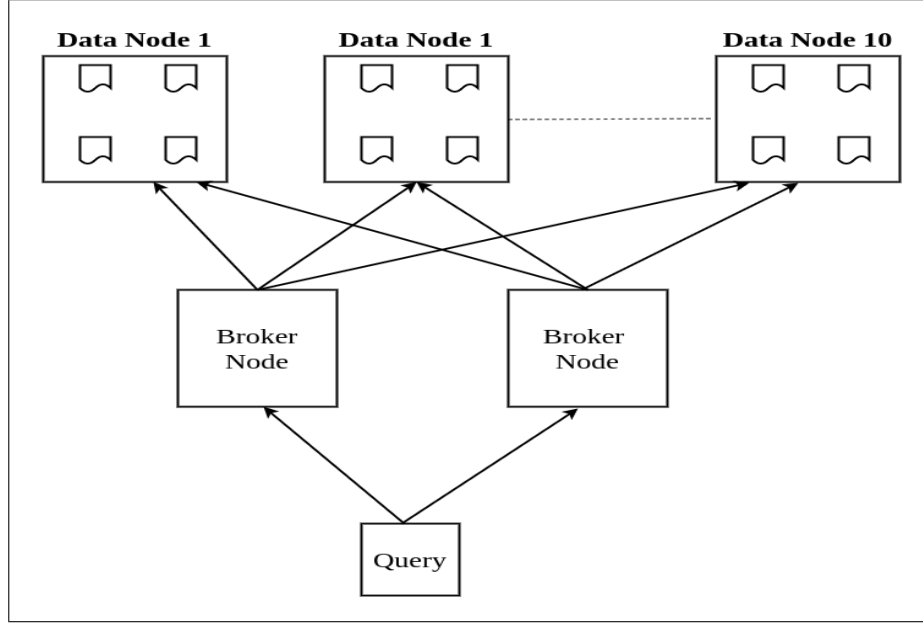


Figure 3.3: System Architecture for the implementation of HOSSA

3.5.2 Algorithm Implementation

As explained above, HOSSA is the combination of three algorithms namely called ReDDE, Sushi and Rank-S as shown in Figure 3.4.

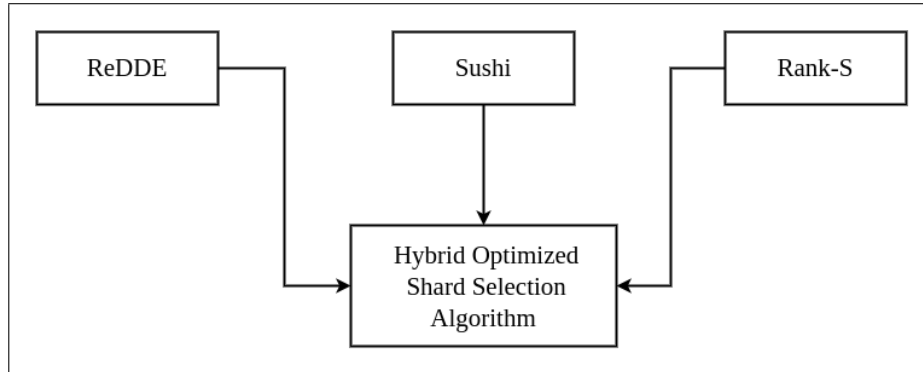


Figure 3.4: Block Diagram showing formation of HOSSA

This new algorithm utilizes the centralized indexing at broker and central ranking approach of ReDDE, Rank Adjustment approach of Sushi, and Voting system approach to select top documents of Rank-S.

Firstly, the centralized sample index(CSI) is formed using the ReDDE approach. Hence, the number of document relevance to form the centralized sampling in the shard S_i is calculated as:

$$Rel(S_{i,q}) = \sum_{d \in S_{i_sampl}} P(Rel|d) \times \frac{|S_i|}{|S_{i_sampl}|} \quad (3.1)$$

$Rel(S_{i,q})$ = Estimated Number of document relevant to query q in shard S_i

q = query in a shards over the documents-collection S_i

$P(Rel|d)$ = The estimated probability of relevance for document d to query q in CCI

S_{i_sampl} = set of sample document from shard S_i

Here, $P(Rel|d)$ is taken from top K initial samples from each shard where K is adjusted experimentally to form the optimum value.

And, the central rank of the document is approximated using the formula:

$$rank_central(d_i) = \sum_{rank_samp(d_j) < rank_samp(d_i)} \frac{|S_j|}{S_{j_sampl}} \quad (3.2)$$

Now, above from above ReDDE formula, SUSHI formula is used for the rank adjustment given as:

$$rank_adjusted(d_{adjusted.i}) = (rank_central(d_i) + 0.5) \times \frac{|c|}{|S_c|} \quad (3.3)$$

Which shows the ratio of the size of the shard they belong to i.e. $|S|$ to the size of the sample from that shard $|S_{sampl}|$.

Now, instead of directly selecting the top K documents from rank, again voting system among the top $(k + \alpha)$ is done using the approach of RANK-S where α is the tunable constant according to the user preferences. A binary tree is built among the selected documents from the bottom-up approach placing the highest

rank document first. The document found in the hierarchy can cast a vote for the shard it belongs to.

$$Vote(d_{adjusted}) = S \times B^{-U} \quad (3.4)$$

S = score of the document given from the CSI ranking

B = exponential base

U = level at which the document was found in the hierarchy

The value of B is experimented and found to be seen stable at the range of 15 to 40.

Based on the voting, the top k documents are selected on priority from $(k + \alpha)$ documents.

The whole process is summarized below:

Hybrid Optimized Shard Selection Algorithm

1: ReDDDe Approach:

a.) Centralized Indexing:

$$Rel(S_{i,q}) = \sum_{d \in s_{i_sampl}} P(Rel|d) \times \frac{|s_i|}{|s_{i_sampl}|}$$

b.) Initial central ranking:

$$rank_central(d_i) = \sum_{rank_saml(d_j) < rank_saml(d_i)} \frac{|S_j|}{S_{j_sampl}}$$

2: SUSHI Approach: Rank Adjustment

$$rank_adjusted(d) = (rank_central(d_i) + 0.5) \times \frac{|c|}{|S_c|}$$

3: RANK-S Approach: Voting to select the top k documents from $(k + \alpha)$ documents

$$Vote(d_{adjusted}) = S \times B^{-U}$$

4: Fetch top K documents obtained from voting and return to the user

Diagrammatically, the above processes are shown in the figure 3.5

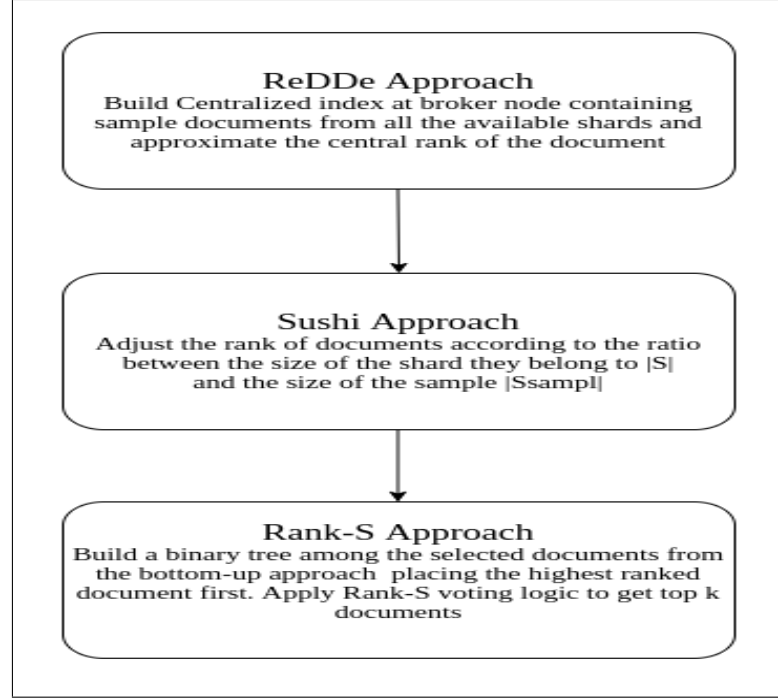


Figure 3.5: Diagram illustrating the core feature of ReDDe, Sushi, and Rank-S extracted for the formation of HOSSA

3.5.3 Tunable shard selection Elasticsearch plugin

As mentioned above, the shard selection process might be based on user requirements (accuracy vs. performance). So, a tunable Elasticsearch plugin is developed by implementing HOSSA. Total 20 shards(primary shards and replica shards) are distributed over 10 nodes for experimentation purpose. The following parameters are tunable:

- i.) Maximum number of shards to be considered for the shard selection. The more the number of shards, the more is the accuracy, but also more latency and vice versa.
- ii.) A tunable constant α considered for voting forming the binary tree. The higher the value of α , more is the accuracy, but also more latency and vice versa.

3.5.4 Special Cases

The plugin skips its operation and allow the default Elasticsearch searching in the following cases:

- When custom **_routing** field is specified in the parameter
- When the ID of the document is stated in the JSON Body

In the above-mentioned cases, Elasticsearch has its internal sophisticated mechanism to select the shard. So, executing the shard selection algorithm isn't required.

3.6 Evaluation Metrics

Following are the evaluation metrics that can be used to evaluate the shard selection algorithms.

3.6.1 Document Score

The document score determines the relevance score i.e how relevance is the search term. Document Score is calculated using the following formula.

$$score(q, d) = queryNorm(q) \cdot coord(q, d) \cdot \sum (tf(t \text{ in } d) \cdot idf(t)^2 \cdot t.getBoost() \cdot norm(t, d))(t \text{ in } q) \quad (3.5)$$

$score(q, d)$ = relevance score for document d on query q

$queryNorm(q)$ = query normalization factor

$coord(q, d)$ = query coordination factor

$tf(t \text{ in } d)$ = term frequency of term t in document d

$idf(t)$ = inverse document frequency for term t

$t.getBoost()$ = the boost applied to the query

$norm(t, d)$ = the field length normalization

Term Frequency (tf) is calculated as:

$$tf(t \text{ in } d) = \sqrt{frequency} \quad (3.6)$$

Inverse Document Frequency (idf) is calculated using the formula:

$$idf = 1 + \ln \frac{numDocs}{docFreq + 1} \quad (3.7)$$

where, \ln = natural logarithm

Field length normalization (norm) is calculated as:

$$norm = \frac{1}{\sqrt{numFieldTerms}} \quad (3.8)$$

Query Normalization Factor(queryNorm) is calculated as:

$$queryNorm = \frac{1}{\sqrt{sumOfSquaredWeights}} \quad (3.9)$$

Finally, the query coordination factor is

$$coord = (term \text{ score}) * \frac{(number \text{ of matching terms})}{(total \text{ number of terms in the query})} \quad (3.10)$$

3.6.2 Search Latency

Search Latency is an important evaluation metric. The main purpose of the shard selection algorithm is to decrease the search latency and get results to the user as fast as possible.

3.6.3 Precision at K (P@K)

A variant of precision called precision at K or P@K can be used which is a standard metric in recent research. The first K documents are considered in this variant while measuring precision. This is because of the assumption that only the top results of the query are relevant to the users.

$$precision = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{relevant\ documents\}|} \quad (3.11)$$

3.6.4 Recall at K (R@K)

A variant of recall called recall at K or R@K can be used which is also a standard metric in recent research. It also considers the first K documents.

$$recall = \frac{|\{relevant\ documents\} \cap \{retrieved\ documents\}|}{|\{retrieved\ documents\}|} \quad (3.12)$$

3.7 Validations

Validation is done on Shakespeare's play dataset. Following is the description of the validation query executed:

Query1:

Query Type: Match Query

Query Index: shakespeare

Field Queried: PlayerLine

Search String: "Did lately meet in the intestine shock"

Query2:

Query Type: Match Query

Query Index: shakespeare

Field Queried: PlayerLine

Search String: "King Henry"

Query3:**Query Type:** Match Query**Query Index:** shakespeare**Field Queried:** PlayerLine**Search String:** “Did lately meet in the intestine shock”

3.8 Experimental Setup

All the experiments were carried out in AWS Elasticsearch servers. Different experimental setups is carried out for experimentation and validation purposes:

For the experimentation purpose, following is the server description:

Table 3.4: AWS Elasticsearch Server specification for the experimentation

Instance Type	vCPU	Memory (GiB)	Number of Data Nodes	Number of Broker Nodes
r5.large.elasticsearch	2	16	10	2

For the validation purpose, following is the server description:

Table 3.5: AWS Elasticsearch Server specification for the validation

Instance Type	vCPU	Memory (GiB)	Number of Data Nodes	Number of Broker Nodes
c6g.large.elasticsearch	2	4	5	1

3.9 Tools and Resources

- IDE(Intelij)
- Programming Language: Java, Spring Boot Framework
- Amazon Elasticsearch Service Instance: r5.large.elasticsearch, c6g.large.elasticsearch

- Elasticsearch, Logstash, Kibana, Filebeat
- Load/API Testings: Jmeter

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Experimental Results and Discussions

As stated in the section 3, total of 15 queries are used for the experimentation. Among them the results for 3 different types of queries and aggregate results of all those 15 queries are shown below.

The search latencies for different algorithms for three types of queries mentioned in the section 3 are shown in figure 4.1, figure 4.2 and figure 4.3.

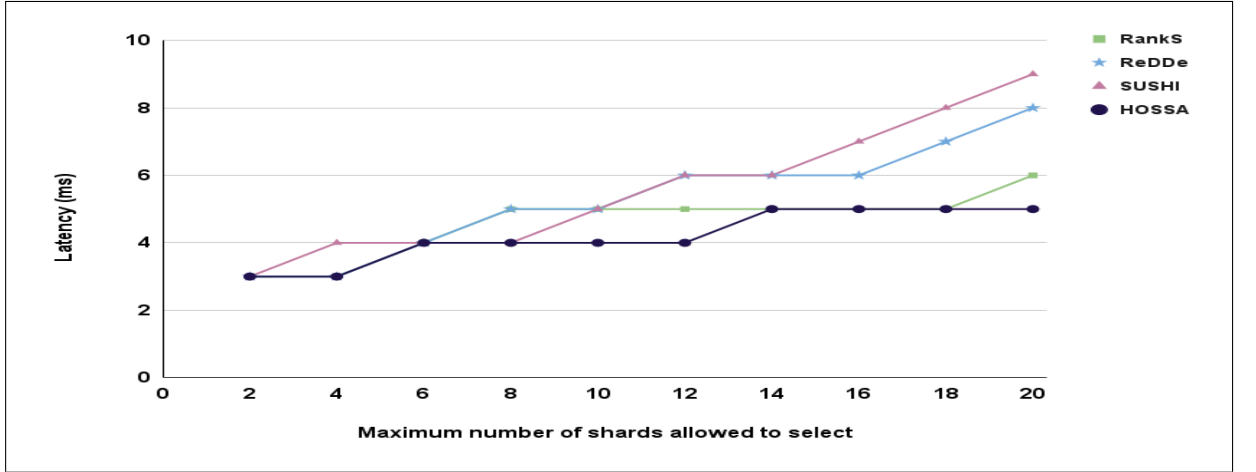


Figure 4.1: Search Latency (ms) against Maximum number of shards allowed to select for different shard selection algorithms on 1st query

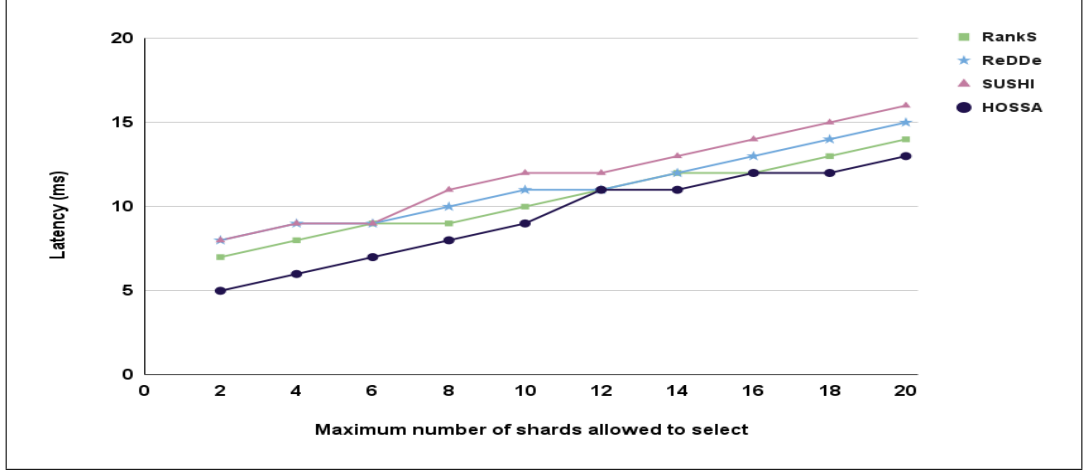


Figure 4.2: Search Latency (ms) against Maximum number of shards allowed to select for different shard selection algorithms on 2nd query

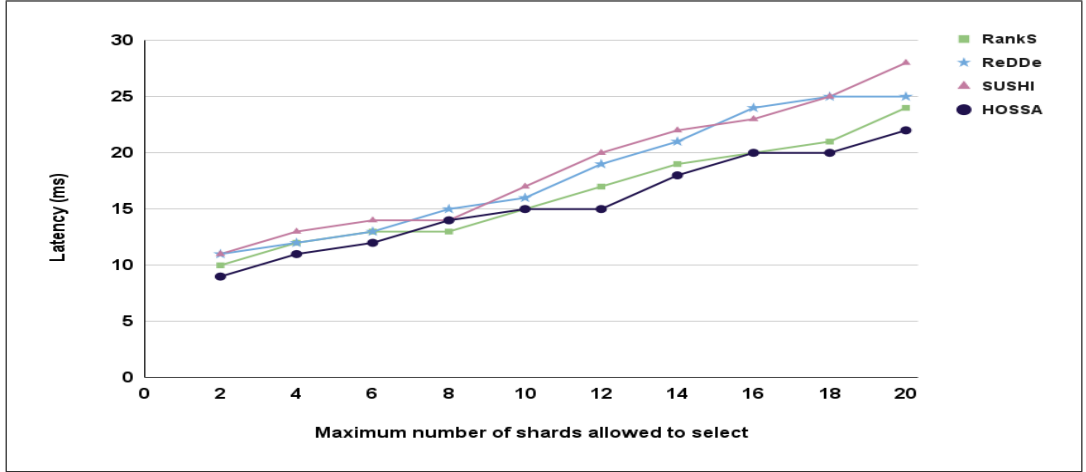


Figure 4.3: Search Latency (ms) against Maximum number of shards allowed to select for different shard selection algorithms on 3rd query

As shown in the above figures, as the maximum number of shards the algorithm allowed to select increased, the search latency is increasing. It is because the broker node has to reach the extra shard distributed over the data nodes. Also, we can see that the HOSSA is performing better in terms of search latency compared to other algorithms. Besides, Rank-S seems to be close to the HOSSA.

The average document score for different algorithms for three types of queries is shown in figure 4.4, figure 4.5 and 4.6.

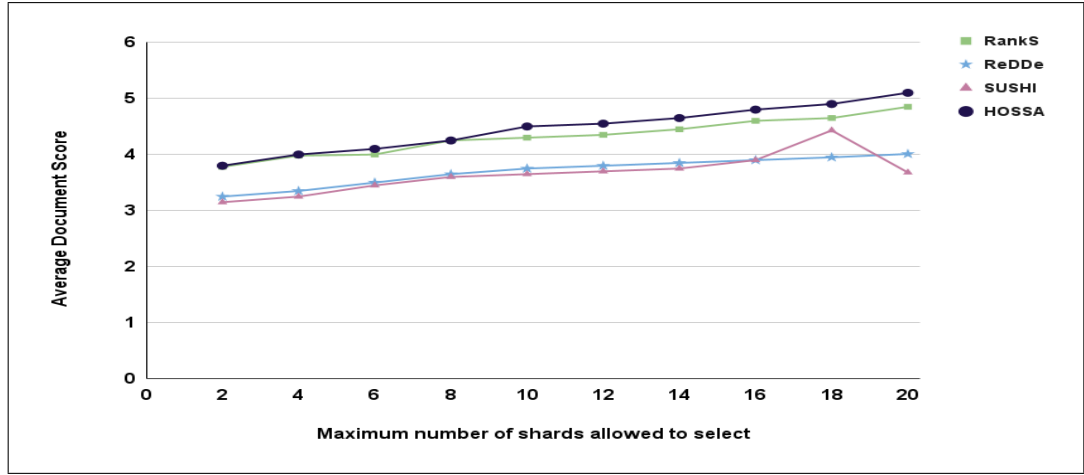


Figure 4.4: Average Document Score against Maximum number of shards allowed to select for different shard selection algorithms on 1st Query

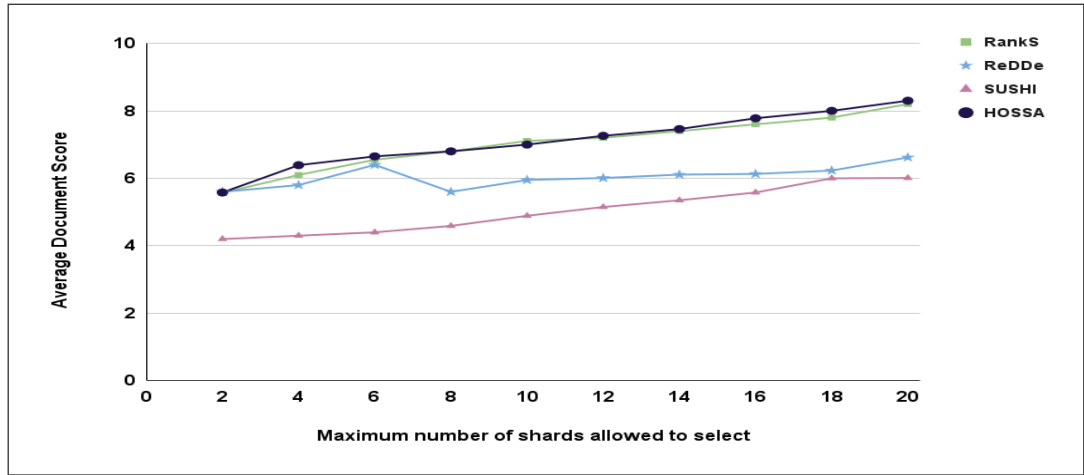


Figure 4.5: Average Document Score against Maximum number of shards allowed to select for different shard selection algorithms for 2nd Query

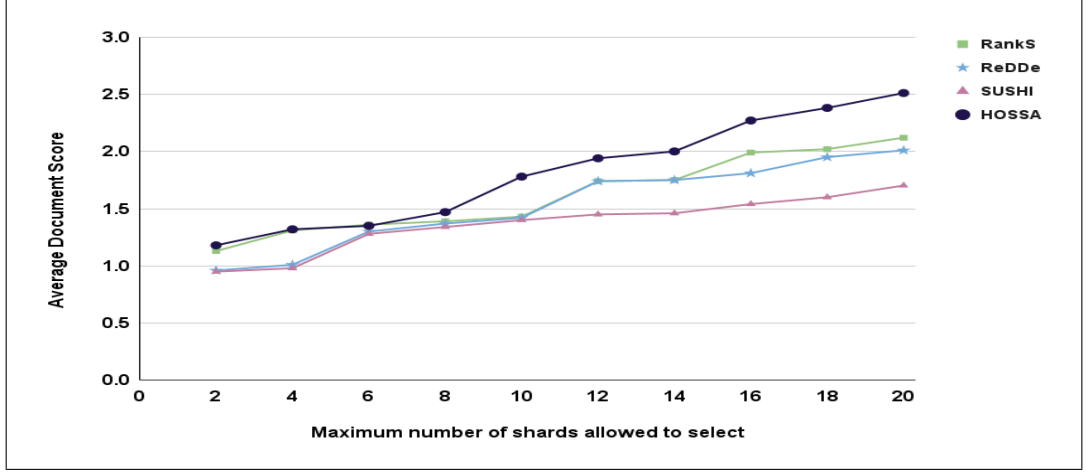


Figure 4.6: Average Document Score against Maximum number of shards allowed to select for different shard selection algorithms for 3rd Query

As shown in the above figures, as the maximum number of shards the algorithm allowed to select increased, the average Document score is increasing. It is because the broker node can select more relevant data from more shards. Also, we can see that the HOSSA is performing better in terms of Average Document Score compared to other algorithms. Besides, Rank-S seems to be close to the HOSSA.

Precision and Recall are calculated in reference to the relevant and retrieved documents in default search when no shard selection algorithm is applied. Precision@10, Precision@30, Recall@10, Recall@30 for different types of queries are shown below.

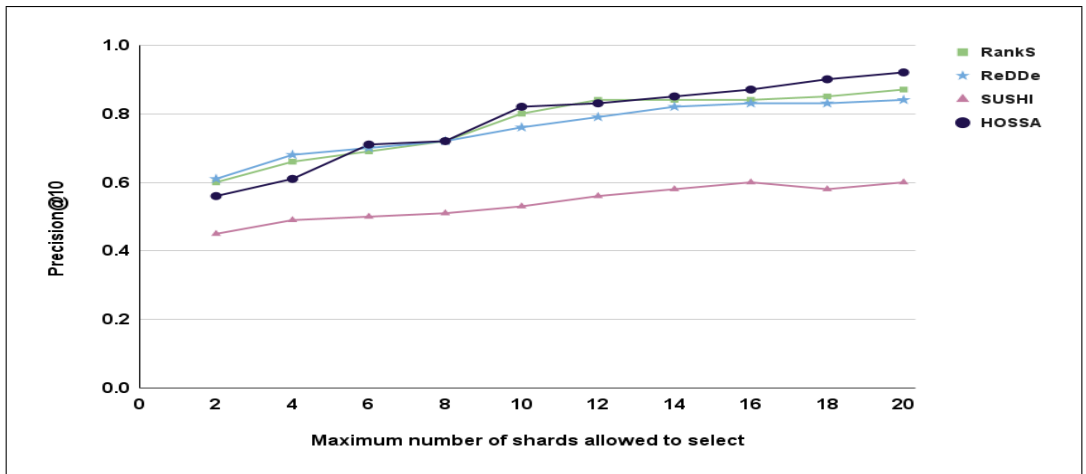


Figure 4.7: Precision@10 against Maximum number of shards allowed to select for different shard selection algorithms for 1st Query

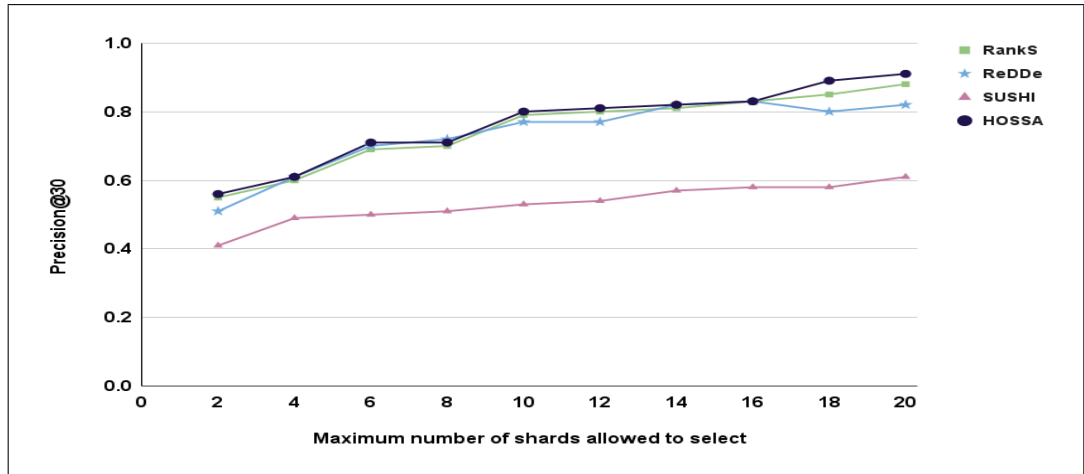


Figure 4.8: Precision@30 against Maximum number of shards allowed to select for different shard selection algorithms for 1st Query

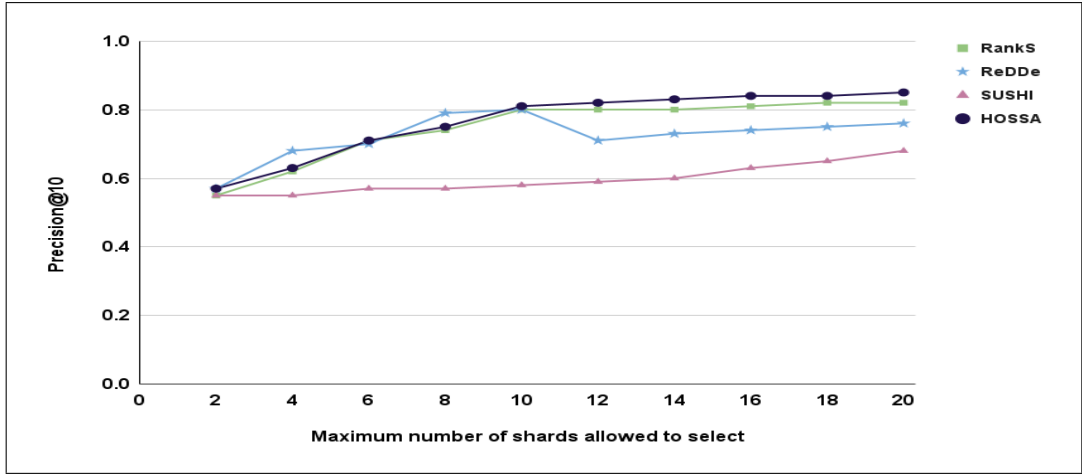


Figure 4.9: Precision@10 against Maximum number of shards allowed to select for different shard selection algorithms for 2nd Query

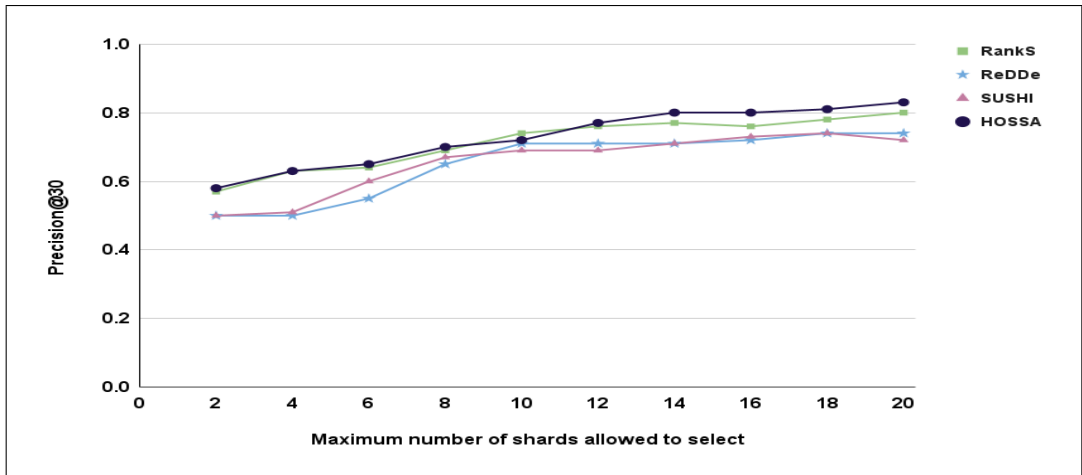


Figure 4.10: Precision@30 against Maximum number of shards allowed to select for different shard selection algorithms for 2nd Query

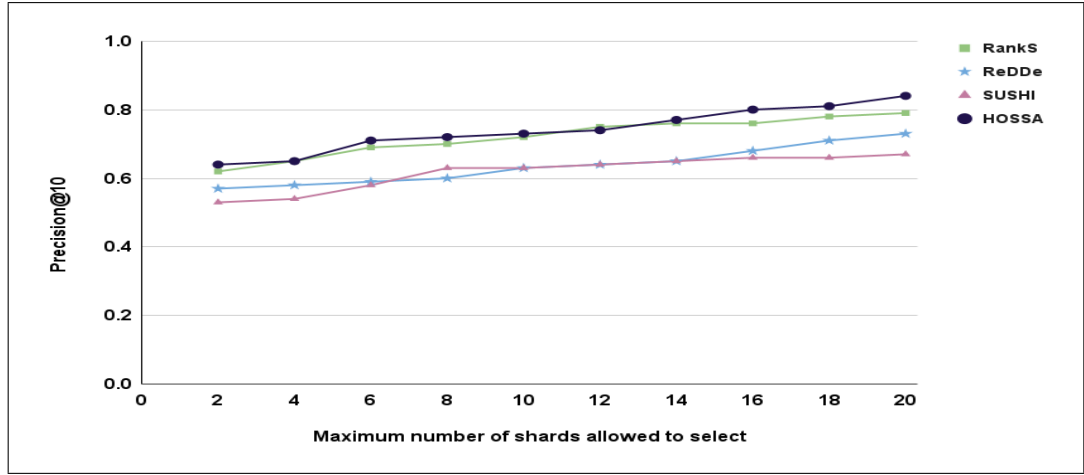


Figure 4.11: Precision@10 against Maximum number of shards allowed to select for different shard selection algorithms for 3rd Query

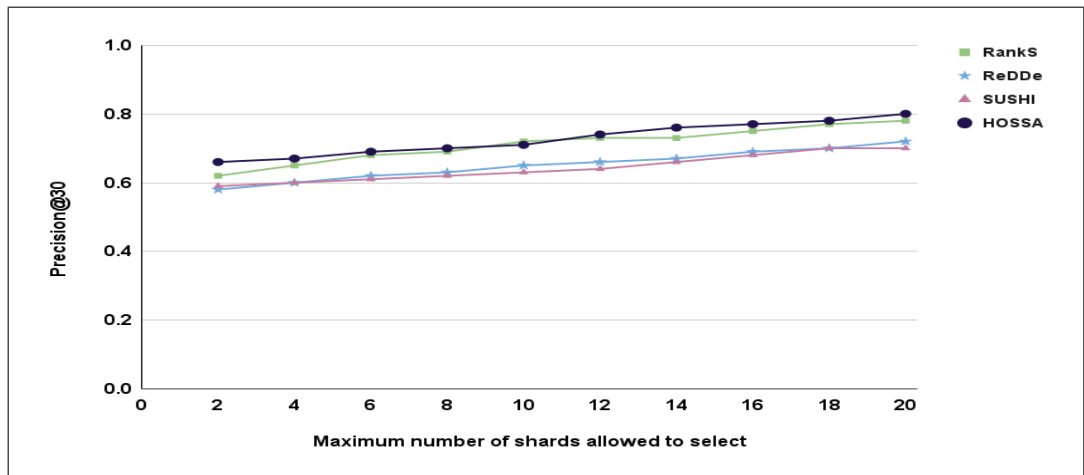


Figure 4.12: Precision@30 against Maximum number of shards allowed to select for different shard selection algorithms for 3rd Query

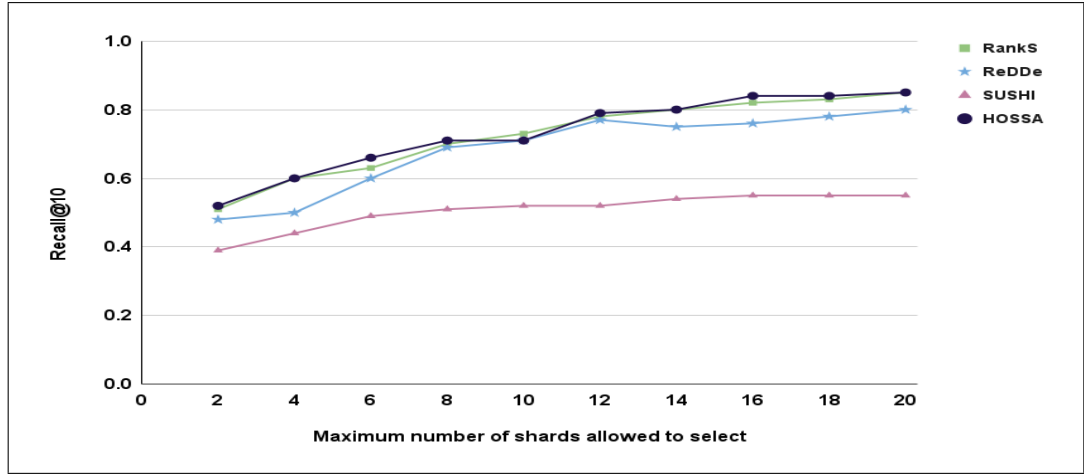


Figure 4.13: Recall@10 against Maximum number of shards allowed to select for different shard selection algorithms for 1st Query

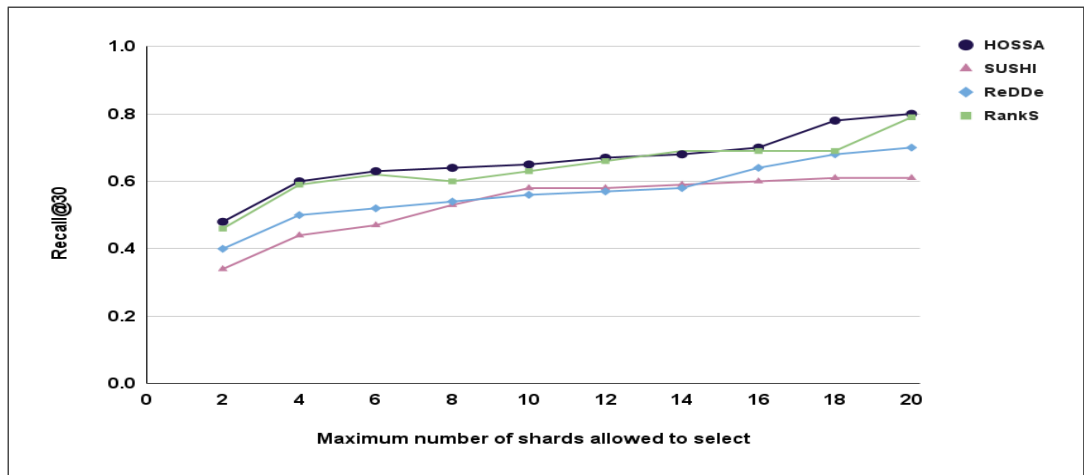


Figure 4.14: Recall@30 against Maximum number of shards allowed to select for different shard selection algorithms for 1st Query

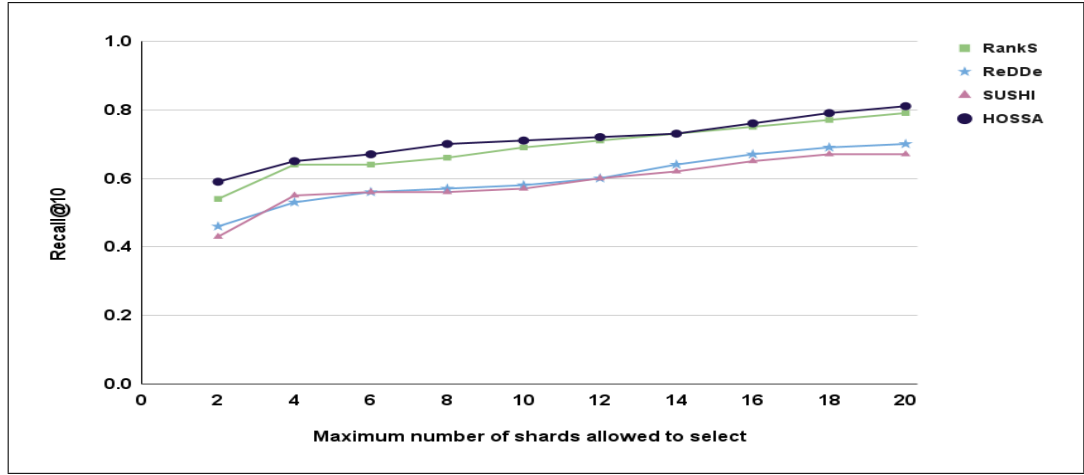


Figure 4.15: Recall@10 against Maximum number of shards allowed to select for different shard selection algorithms for 2nd Query

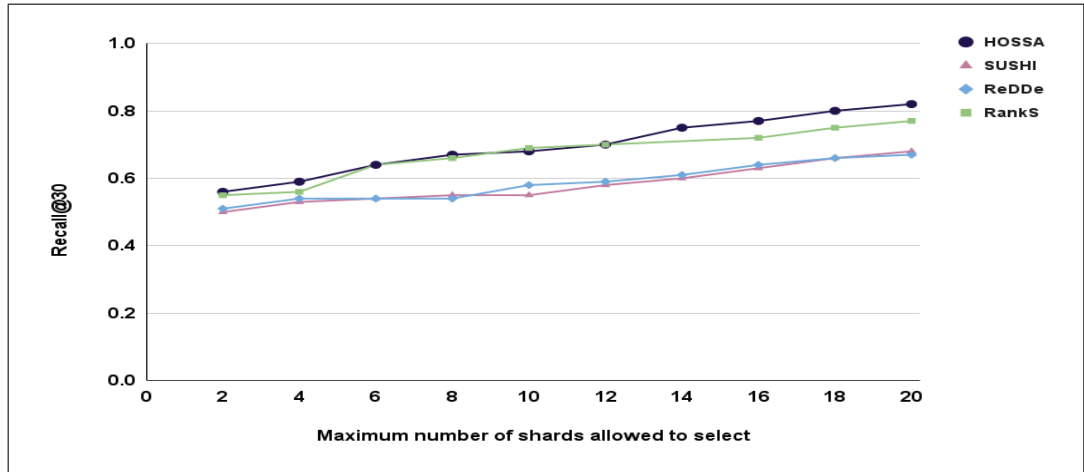


Figure 4.16: Recall@30 against Maximum number of shards allowed to select for different shard selection algorithms for 2nd Query

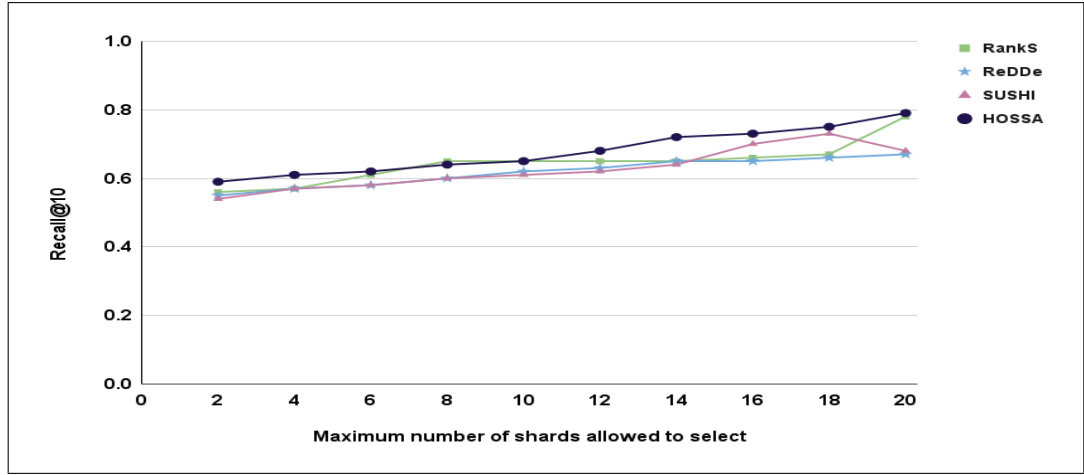


Figure 4.17: Recall@10 against Maximum number of shards allowed to select for different shard selection algorithms for 3rd Query

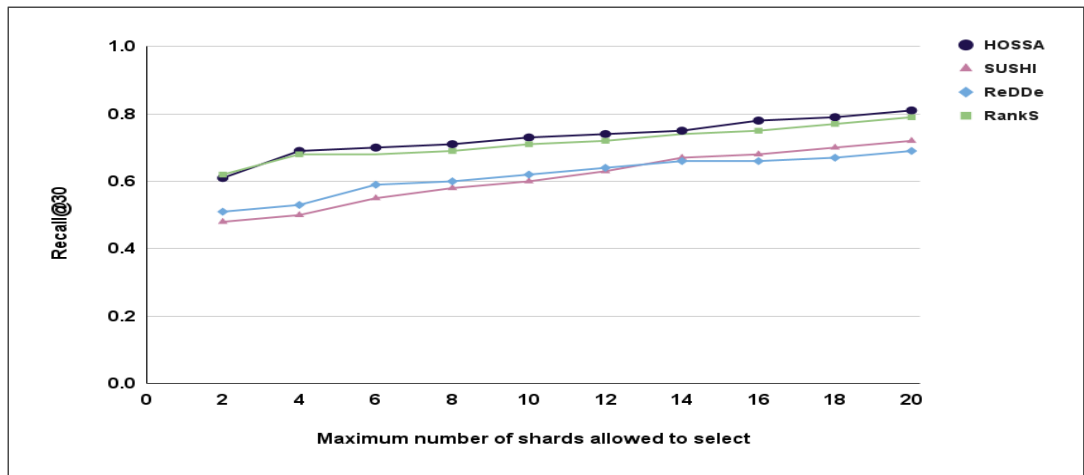


Figure 4.18: Recall@30 against Maximum number of shards allowed to select for different shard selection algorithms for 3rd Query

For the Precision@10 and Recall@10, the top 10 documents are selected whereas for Precision@30 and Recall@30, the top 30 documents are considered. This works well because the users are mostly interested in top results from the query search. As shown in the above figures, the recall and precision are improving as the maximum number of shards the algorithm allowed to select increased. It is because the broker node can get more relevant documents from more shards. Also, we can see that the HOSSA is performing better in terms of Average Document Score compared to other algorithms. Besides, Rank-S seems to be close to the HOSSA.

On some of the instances in the above results, Rank-S seems to be performing the same or better than the HOSSA mainly when the maximum number of shards allowed to select is lower. Also, other algorithms are approaching near the HOSSA when the maximum number of shards allowed to select is lower. This is due to the fact that the HOSSA is designed to work better in the higher number of shards, which is the motive of the Shard Selection Algorithm as well.

The following table shows the average value obtained performing the search operations using 15 queries when the maximum shard allowed to select ranges from 2 to 20.

Table 4.1: Table showing average value of the experimentation results obtained from 15 queries

Shard Selec- tion Meth- ods	Search Latency (ms)	Average Docu- ment Score	P@10	P@30	R@10	R@30
RankS	17	5.39	0.73	0.71	0.68	0.66
ReDDe	18	4.53	0.68	0.66	0.62	0.56
SUSHI	19	3.73	0.52	0.57	0.54	0.54
HOSSA	16	5.58	0.74	0.73	0.70	0.68

A tunable constant α considered for voting forming the binary tree form the relationship with Search Latency and Average Document Score as shown in figure 4.19 and 4.20 respectively.

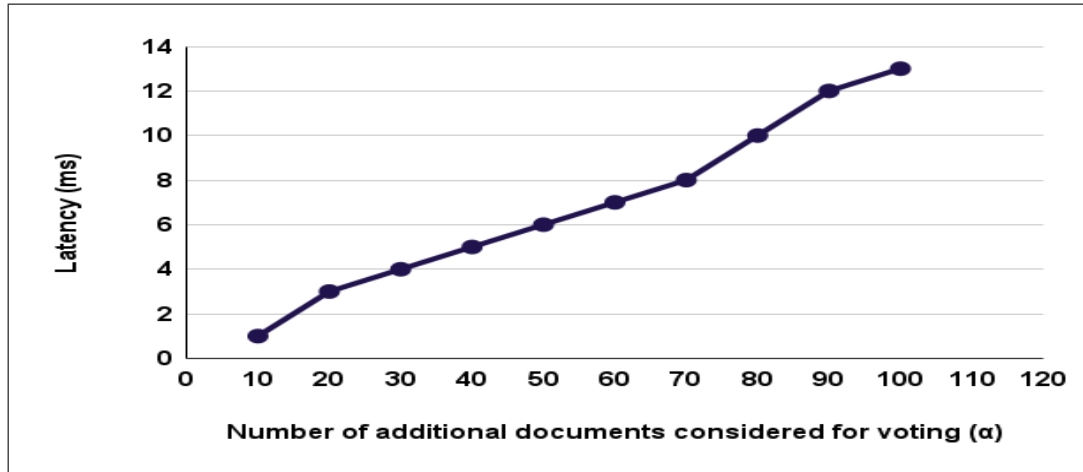


Figure 4.19: Search Latency (ms) against α , a tunable constant considered for voting

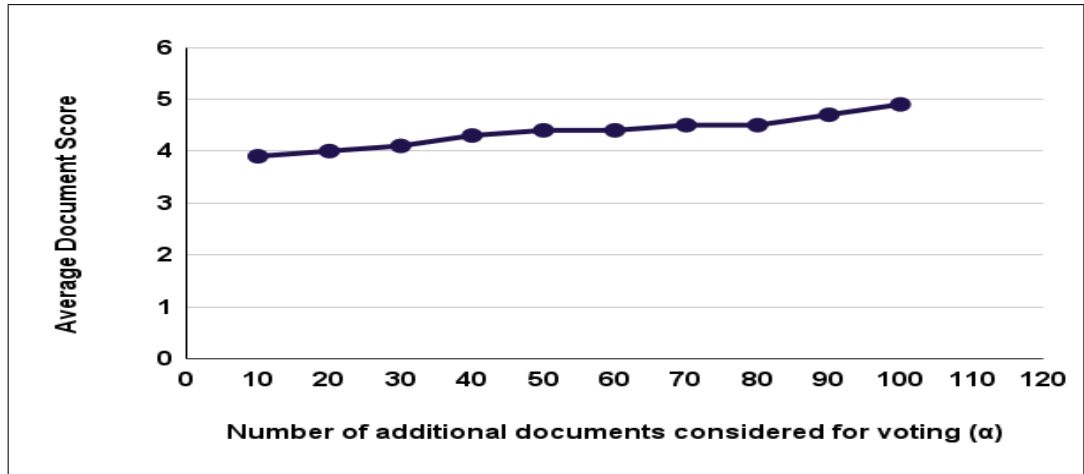


Figure 4.20: Average Document Score against α , a tunable constant considered for voting

As shown in the above figure, the Higher the value of α , the more is the accuracy, but also more search latency and vice versa.

The correlation for the Search Latency and Heap Memory for a node is shown in the figure 4.21.

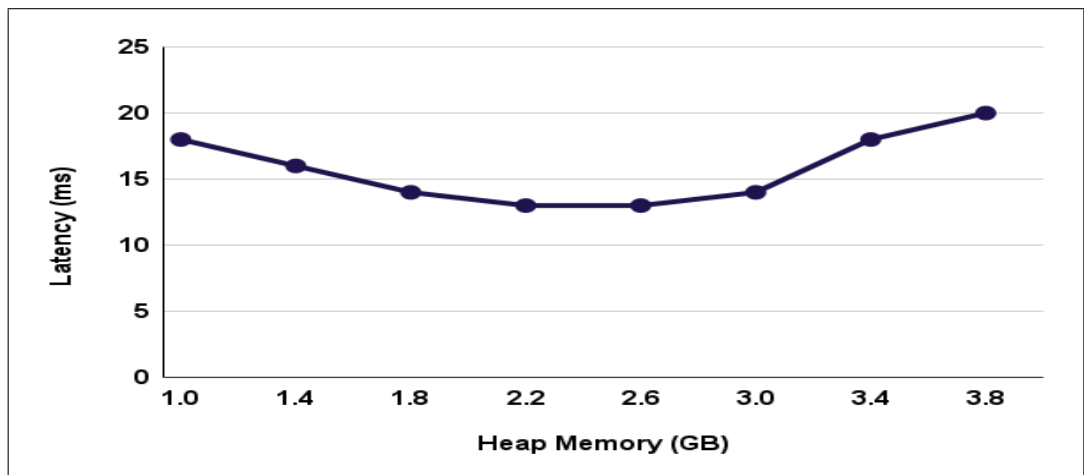


Figure 4.21: Search Latency(ms) against Heap Memory(GB)

As shown in the above figure, increment of Heap memory decrease the search latency up to a certain limit. Heap memory above certain increases search latency drastically. Also, the heap memory allocation depends largely on required operations.

4.2 Validation Results

The following table shows the average value of the validation results obtained performing the search operations using three queries described in the methodology section. The validation is done in a different set of data, queries, and server configuration than that of the experimentation.

Table 4.2: Table showing average value of the validation results obtained from three queries

Shard Selection Methods	Search Latency (ms)	Average Document Score	P@10	P@30	R@10	R@30
RankS	23	4.32	0.74	0.72	0.57	0.67
ReDDe	25	3.45	0.59	0.62	0.59	0.57
SUSHI	26	3.65	0.61	0.65	0.62	0.59
HOSSA	20	4.46	0.76	0.74	0.71	0.70

As shown in the above table, HOSSA is performing better compared to other algorithms in the validation data as well.

CHAPTER 5

CONCLUSION AND FUTURE ENHANCEMENT

5.1 Conclusion

A new shard selection algorithm called Hybrid Optimized Shard Selection Algorithm is developed using core features of existing popular shard selection algorithms called ReDDe, SUSHI, and Rank-S. For that, the Elasticsearch platform is chosen for the implementation.

As already mentioned, we have divided the data into 20 primary shards and replica shards. From those 20 shards, the maximum number of shards allowed to select is tunable. As shown from the above results, more the number of shards allowed to select more is the Document Score, Precision, and Recall at the cost of the more Latency. This is because the broker node has to work on extra shard gaining extra information about a given query at the cost of more time and resources.

As we can see from the above experiments, in some of the instances, Rank-S seems to be performing the same or better than HOSSA mainly when the maximum number of shards allowed to select is lower. Also, other algorithms are approaching HOSSA when the maximum number of shards allowed to select is lower. This is due to the fact that the HOSSA is designed to work better in a higher number of shards. This is totally valid because we mainly require Shard Selection Algorithm to be applied when the number of shards is high for the performance improvements.

From the above experiments, in terms of average latency, considering overall queries and the maximum number of shards allowed to select, the HOSSA is performing 19.34%, 15.6%, and 7.30% better than SUSHI, ReDDe, and Rank-S respectively.

In terms of Average Document Score, considering overall queries and the maximum number of shards allowed to select, the HOSSA is performing 33.09%, 18.89%, and 3.31% better than SUSHI, ReDDe, and Rank-S respectively.

In terms of Precision@10, considering overall queries and a maximum number of shards allowed to select, the HOSSA is performing 30.25%, 7.93%, and 2.00% better than SUSHI, ReDDe, and Rank-S respectively.

In terms of Precision@30, considering overall queries and the maximum number of shards allowed to select, the HOSSA is performing 21.47%, 8.97%, and 2.21% better than SUSHI, ReDDe, and Rank-S respectively.

In terms of Recall@10, considering overall queries and the maximum number of shards allowed to select, the HOSSA is performing 23.34%, 11.67%, and 3.06% better than SUSHI, ReDDe, and Rank-S respectively.

In terms of Recall@30, considering overall queries and the maximum number of shards allowed to select, the HOSSA is performing 21.00%, 17.94%, and 2.95% better than SUSHI, ReDDe, and Rank-S respectively.

The relationship of latency and average document score with a tunable constant α considered for voting is established. The higher the value of α , the more is the accuracy, but also the more latency and vice versa.

Also, a relationship between heap memory and latency is established. An increment of Heap memory decreases the latency up to a certain limit. Heap memory above certain increases latency drastically. Also, the heap memory allocation depends largely on required operations.

The validation is done in a different set of data, queries, and server configuration than that of the experimentation. HOSSA is performing better compared to other

algorithms in the validation data as well.

5.2 Future Works

There is some room for future enhancement in the research. The shard Selection plugin developed for the thesis is Elasticsearch specific. A modification to the plugin can be done to make it generic so that it can be used on other distributed platforms like Solr and MongoDB. Also, currently, as users can tune two parameters based on their preference for relevance or performance, research on more such user tunable parameters can be done.

REFERENCES

- [1] Zhuyun Dai, Chenyan Xiong, and Jamie Callan. Query-biased partitioning for selective search. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, pages 1119–1128, 2016.
- [2] Hrishikesh Karambelkar. *Scaling Big Data with Hadoop and Solr*. Packt Publishing Ltd, 2013.
- [3] Mohamad Sobhie. Tuning of elasticsearch configuration-parameter optimization through simultaneous perturbation stochastic approximation algorithm. Master’s thesis, 2019.
- [4] Quentin Coviaux. Optimization of the search engine elasticsearch. Master’s thesis, Universitat Politècnica de Catalunya, 2019.
- [5] Rafal Kuc and Marek Rogozinski. *Elasticsearch server*. Packt Publishing Ltd, 2013.
- [6] James P Callan, Zhihong Lu, and W Bruce Croft. Searching distributed collections with inference networks. In *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 21–28, 1995.
- [7] Daryl D’Souza, James A Thom, and Justin Zobel. Collection selection for managed distributed document databases. *Information processing & management*, 40(3):527–546, 2004.
- [8] Milad Shokouhi and Luo Si. Federated search”. foundations and trends in information retrieval (ftir). *Foundations and Trends in Information Retrieval*, 2011.
- [9] Luo Si and Jamie Callan. Relevant document distribution estimation method for resource selection. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 298–305, 2003.

- [10] Paul Thomas and Milad Shokouhi. Sushi: Scoring scaled samples for server selection. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 419–426, 2009.
- [11] Anagha Kulkarni, Almer S Tigelaar, Djoerd Hiemstra, and Jamie Callan. Shard ranking and cutoff estimation for topically partitioned collections. In *Proceedings of the 21st ACM international conference on information and knowledge management*, pages 555–564, 2012.
- [12] Praveen M Dhulavvagol, Vijayakumar H Bhajantri, and SG Totad. Performance analysis of distributed processing system using shard selection techniques on elasticsearch. *Procedia Computer Science*, 167:1626–1635, 2020.
- [13] Per Berglund. Shard selection in distributed collaborative search engines a design, implementation and evaluation of shard selection in elasticsearch. 2014.
- [14] Ilya Markov and Fabio Crestani. Theoretical, qualitative, and quantitative analyses of small-document approaches to resource selection. *ACM Transactions on Information Systems (TOIS)*, 32(2):1–37, 2014.
- [15] Zhanglong Wang and Yang Pi. An optimization strategy of shard on elasticsearch.
- [16] Yimeng Liu, Yizhi Wang, and Yi Jin. Research on the improvement of mongodb auto-sharding in cloud environment. In *2012 7th international conference on Computer science & education (ICCSE)*, pages 851–854. IEEE, 2012.

APPENDIX A

Queries Used in the experimentation

Descriptions of 15 Queries used for the experimentation are given below:

Query1:

Query Type: Match Query

Query Index: http

Field Queried: content

Search String: “players coaches”

Query2:

Query Type: Match Query

Query Index: email

Field Queried: message

Search String: “greeting from here”

Query3:

Query Type: Match Query

Query Index: email

Field Queried: message

Search String: “hello world”

Query4:

Query Type: Match Query

Query Index: http

Field Queried: content

Search String: “Fredy Montero”

Query5:

Query Type: Match Query

Query Index: http

Field Queried: content

Search String: “reluctant to welcome German athletes”

Query6:

Query Type: Match Query

Query Index: http

Field Queried: content

Search String: “Eastern Christianity”

Query7:

Query Type: Match Query

Query Index: http

Field Queried: content

Search String: “Western Christianity”

Query8:

Query Type: Match Query

Query Index: http

Field Queried: content

Search String: “The Seattle Times”

Query9:

Query Type: Match Query

Query Index: http

Field Queried: content

Search String: “Mustang”

Query10:

Query Type: Match Query

Query Index: email

Field Queried: message

Search String: “Published by Scholastic”

Query11:

Query Type: Match Query

Query Index: email

Field Queried: message

Search String: “Hello readers”

Query12:

Query Type: Match Query

Query Index: email

Field Queried: message

Search String: “regards”

Query13:

Query Type: Match Query

Query Index: email

Field Queried: message

Search String: “thank you”

Query14:

Query Type: Match Query

Query Index: email

Field Queried: message

Search String: “yours sincerely”

Query15:

Query Type: Match Query

Query Index: email

Field Queried: message

Search String: “attached”

APPENDIX B

Turnitin Similarity Index

MSCKE_Final_Thesis_Report_for_SI/075mscke_020_Yashasv...

ORIGINALITY REPORT

16%
SIMILARITY INDEX

PRIMARY SOURCES

1	hdl.handle.net Internet	307 words — 4%
2	flipkarma.com Internet	300 words — 4%
3	citeseerx.ist.psu.edu Internet	94 words — 1%
4	www.irjet.net Internet	48 words — 1%
5	findwise.com Internet	43 words — 1%
6	usermanual.wiki Internet	40 words — 1%
7	eprints.utm.edu.my Internet	32 words — < 1%
8	"Encyclopedia of Database Systems", Springer Science and Business Media LLC, 2018 Crossref	22 words — < 1%
9	Praveen M Dhulavvagol, Vijayakumar H Bhajantri, S G Totad. "Performance Analysis of Distributed	21 words — < 1%