Performance Enhancement of Distributed Processing Systems using Novel Hybrid Shard Selection Algorithm

Praveen M. Dhulavvagol
School of Computer Science and Engineering, KLE Technological University, India
praveen.md@kletech.ac.in (corresponding author)

Sashikumar G. Totad
School of Computer Science and Engineering, KLE Technological University, India
totad@kletech.ac.in

Received: 22 February 2022 | Revised: 7 March 2024 | Accepted: 12 March 2024

 DOI: https://doi.org/10.48084/etasr.7128

ABSTRACT

Distributed processing systems play a crucial role in query search operations, where large-scale data are partitioned across multiple nodes using shard selection algorithms. However, the existing shard selection algorithms pose significant challenges, such as shard ranking, shard cut-off estimation, high latency, low throughput, and high processing costs. These limitations become more pronounced as the data size increases, affecting the efficiency and effectiveness of search operations. To address these challenges, the novel Hybrid Shard Selection Algorithm (HSSA) is proposed as a solution in this paper, designed specifically to enhance the effectiveness and efficiency of search operations within distributed processing systems. HSSA employs an advanced sharding approach that adeptly navigates and targets pertinent shards based on specific queries. This not only curtails search-related overhead but also enhances operational efficiency. Through rigorous testing using the Gov2 dataset, the HSSA algorithm has proven its merits. When set against well-established algorithms like CORI, Rank-S, and SHiRE, HSSA stands out, registering remarkable gains in average throughput by 21%, 16%, and 12%, while also slashing latency by 14.2%, 9.4%, and 8.2%, respectively. The insights gained from this research underscore HSSA's capability to effectively bridge the gaps inherent in traditional shard selection strategies. Furthermore, its exemplary efficacy with datasets of varied sizes amplifies its relevance for practical integration within distributed processing landscapes.

Keywords-sharding; cluster; indexing; partitioning; allocation

I. INTRODUCTION

The exponential growth of data across industries and disciplines has prompted a paradigm shift in data processing, storage, and retrieval methods. Distributed processing systems have emerged as a dominant paradigm in the era of big data, enabling the efficient processing and analysis of vast amounts of information. The cornerstone of these systems lies in their ability to divide data and tasks across multiple nodes, ensuring parallelism, scalability, and fault tolerance [1]. A critical component in optimizing the performance of such systems is sharding, which entails partitioning databases into smaller, more digestible pieces, termed as shards, which can be processed concurrently. Shard selection essentially involves the partitioning of a database or dataset into smaller, more manageable segments, known as shards. Each shard can then be processed independently and concurrently across various nodes of the distributed system. This concurrent processing not only accelerates data retrieval and computation, but also ensures a balanced load distribution across the system, preventing potential bottlenecks and system overloads. Traditional sharding approaches have predominantly revolved around either static or dynamic methodologies. Static sharding involves dividing a database based on predetermined criteria, which, while simple and predictable, may lead to inefficiencies such as data skew or imbalance among nodes. On the other hand, dynamic sharding techniques allow the system to partition data on-the-fly based on various parameters, such as data size or workload characteristics. While more flexible, dynamic sharding can sometimes add overhead due to the continuous monitoring and adjustment requirements.

Static shard selection algorithms, like the Lexical SHiRE (Lex-S) and Connected SHiRE (Conn-S), are deterministic in nature. Their modus operandi focuses on predefined criteria set during data ingestion or index creation. For instance, Lex-S relies on the lexical attributes of data for sharding, and once formed, these shards remain unchanged irrespective of the nature of queries they confront. Similarly, Conn-S capitalizes on the intrinsic relationships or connected components within

the data to form shards, retaining their structure throughout their lifecycle [2].

In contrast, dynamic shard selection algorithms adjust and evolve based on the incoming queries, offering a more adaptive approach to shard selection. Algorithms such as Rank SHiRE (Rank-S), Collection Retrieval Inference Network (CORI), and Relevant Document Distribution Estimation (ReDDE) epitomize this category. Rank-S, for instance, dynamically ranks shards vis-à-vis their relevance to an incoming query. CORI and ReDDE, on the other hand, estimate the significance of shards based on varying parameters, like document statistics or the relevance of terms in a query, respectively [3]. While both static and dynamic sharding techniques come with their own set of merits, neither is devoid of shortcomings. Static algorithms might be compromised by unforeseen query patterns, whereas dynamic ones may incur overheads due to their adaptive nature [4]. In the wake of these challenges, there's a growing need for an algorithmic solution that seamlessly blends the predictability of static sharding with the adaptability of dynamic sharding [5].

This paper proposes the Hybrid Shard Selection Algorithm (HSSA), a novel approach designed to enhance the performance of distributed processing systems. By intelligently leveraging the strengths of both static and dynamic sharding techniques, HSSA seeks to integrate the robustness of static algorithms with the adaptability of the dynamic ones. By initially leveraging the likes of Lex-S or Conn-S for shard creation and subsequently resorting to dynamic algorithms like Rank-S, CORI, or ReDDE for shard selection during queries, HSSA aspires to strike a harmonious balance. Such an approach promises to fuse the predictability of static sharding with the nimbleness of dynamic sharding, paving the path for a more refined, efficient, and performance-enhanced distributed processing system. HSSA aims to ensure optimal data distribution, minimize processing latency, and promote balanced resource utilization across the system. The main contributions of the proposed algorithm are:

- **Hybrid Integration:** HSSA uniquely combines both static (Lex-S, Conn-S) and dynamic (Rank-S, CORI, ReDDE) shard selection methodologies, optimizing efficiency and adaptability in distributed systems.
- **Balanced Efficiency:** By leveraging static predictability and dynamic real-time relevance, HSSA ensures swift data retrieval and optimal resource utilization.
- **Scalability and Adaptability:** HSSA exhibits enhanced scalability and fault tolerance, adjusting seamlessly to diverse query patterns, and setting a benchmark for future shard selection algorithms.

The primary contribution of HSSA lies in its innovative approach of combining the benefits of both static and dynamic shard selection methodologies to presents a holistic solution. This hybrid methodology aims to ensure optimal data distribution, reduce processing latency, and guarantee efficient resource utilization across the distributed processing system. In essence, the HSSA algorithm sets a new benchmark for shard selection by harmoniously merging predictability with flexibility, ultimately enhancing the overall performance of distributed data processing systems.

II. RELATED WORK

Sharding is a horizontal data partitioning technique that partitions data row-wise into *n* different shards. Authors in [6] focus on enhancing the effectiveness of data reclamation and processing in Elasticsearch through colorful shard selection ways. The study delves into the complications of Elasticsearch's armature, emphasizing the pivotal part of sharding in vertical scaling. The authors explore and compare different shard selection strategies, similar to the range-grounded, hash-grounded, and adaptive ways, assessing their effectiveness in optimizing query prosecution and cargo balancing. The optimization of data partitioning in a sharding system employing harmonious mincing is presented in [2]. The central focus lies in streamlining the process of dividing and distributing data across shards to enhance system effectiveness. The authors address the complications associated with traditional data partitioning styles and propose a simplified approach, using harmonious mincing as the underpinning mechanism. The study delves into the complications of harmonious mincing, emphasizing its part in maintaining cargo balance and easing scalable data storehouses. Authors in [7] report that, the basic CORI algorithm does not scale up with large-size data and downgrades the performance of distributed search engines. In [8], it was reported that selective search could divide the data according to topic-based shards and search could be conducted in only a few shards that contain the relevant query. The experimental results showed that the selective search is more effective than the (traditional) exhaustive search method. Dividing the data according to the topic-based motivates the cluster to search the data for the query in a few shards which contain the relevant data [9]. If the documents have similarities, then they can be grouped. In [1], a novel query-biased partitioning strategy, aligning document partitions with topics derived from query logs is presented. It specifically addresses clustering initialization and document similarity calculation.

A query-driven clustering initialization algorithm employs topics from query logs, while a query-biased similarity metric prioritizes terms important in query logs. Both methods enhance retrieval effectiveness, diminish variance, and promote a more balanced distribution of shard sizes. The SHARD Triple-Store, a cutting-edge, high performance, and massively scalable distributed system leveraging the MapReduce software framework, is presented in [10]. This innovative approach capitalizes on the inherent capabilities of MapReduce to handle vast amounts of data in a parallel and distributed manner. The triple-store architecture is specifically designed for efficient and scalable storage and retrieval of RDF (Resource Description Framework) triples, a crucial component in semantic data processing. By utilizing the MapReduce paradigm, the SHARD Triple-Store achieves exceptional scalability, allowing it to seamlessly handle large-scale datasets. MapReduce's parallel processing capabilities are harnessed to distribute the computational load across a multitude of nodes, ensuring efficient data processing and storage. This architecture excels in scenarios where traditional

relational databases may struggle to cope with the sheer volume and complexity of data. The SHARD Triple-Store represents a notable advancement in the field of distributed systems, offering a robust solution for organizations dealing with massive and dynamic datasets in the realm of semantic data processing.

Significant challenges in distributed processing are performance enhancement, fault tolerance, and scalability [11, 12]. Different approaches and heuristic techniques have been proposed to partition data and allocate shards at multiple nodes. The existing data partitioning strategies encounter high communication, low throughput, scalability, consistency, and mining overhead. The proposed approach aims to overcome the above limitations using a hybrid shard selection technique.

III. HSSA SYSTEM ARCHITECTURE

Figure 1 depicts the proposed HSSA system's architecture. The architecture offers a sophisticated solution to the challenges inherent in managing vast datasets in distributed processing systems. Central to the architecture, is a distributed framework anchored by the Master Node, a pivotal component that supervises the data's initial processing. With the aid of sharding techniques, the master node meticulously segments the massive volume of data into strategically defined shards. Each shard is intricately themed around specific topics, ensuring that every fragment or shard is a coherent aggregation of documents that share thematic resonance. As the orchestrator, the master node oversees operations across the distributed spectrum, managing queries and the critical dual processes of data partitioning and shard selection. This pivotal node's proactive role extends further; by controlling parallel processing across shards, it ensures a dramatic reduction in latency, simultaneously amplifying throughput and refining search precision.

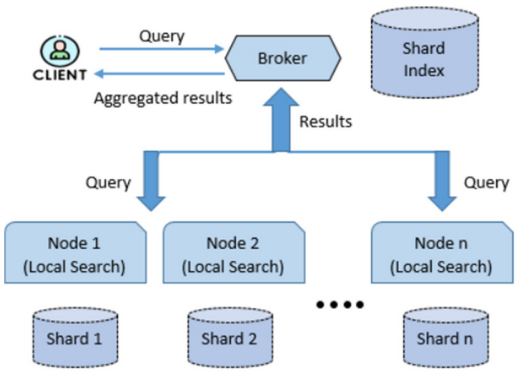

Fig. 1. HSSA system architecture.

Complementing the master node are the Data Nodes, the repositories entrusted with housing the carefully curated shards. Each data node plays steward to a shard—a self-sustaining subset of the entire dataset, a thematic archive of documents under its domain. With HSSA's architectural design, the process is seamlessly streamlined into two core phases. First, the data undergoe a rigorous partitioning process, ensuring it is dissected into topic-centric, logical shards. Following this, the algorithm pivots to the shard selection and ranking phase, pinpointing the shards most relevant to a given query and ranking them in order of relevance. This fusion of methodological rigor and precision-driven selection gives HSSA its distinctive edge, setting new benchmarks in distributed data processing efficiency.

A. Hybrid Data Partitioning Approach in HSSA

Hybrid data partitioning, as encapsulated in HSSA, is an innovative fusion of horizontal and vertical partitioning strategies, offering a flexible and adaptive system for diverse datasets. This approach emerges from the understanding that the strengths of one partitioning type can compensate for the shortcomings of the other, thus ensuring a holistic data management system. The procedure initiates with a Preliminary Data Analysis, where the data's structure, volume, and access patterns are meticulously assessed. By probing into the unique intricacies of the dataset, HSSA can tailor its partitioning mechanisms to match the data's inherent needs.

The horizontal partitioning strategy then comes into play, slicing data at the row level. Each shard, or partition, is structured to represent a consistent portion of the overarching dataset. This approach is instrumental when systems foresee queries that span multiple columns but are concentrated on specific rows. For example, in a dataset profiling users, one might seek all data attributes (columns) for a specific user (a particular row).

Vertical partitioning focuses on column-based data segregation. By clustering columns that are frequently accessed in tandem, this approach optimizes query responses for those seeking specific data attributes across the board. For instance, in an e-commerce database, queries might predominantly target product names and prices, while seldom accessing their detailed descriptions. In such scenarios, vertical partitioning optimizes data fetch times by co-locating these commonly accessed columns.

HSSA's approach is its dynamic melding of these two strategies. The system constantly evaluates query demands and data access patterns, fine-tuning the balance between horizontal and vertical partitions. If a dataset witnesses evolving access patterns, HSSA's dynamic adaptation mechanism adjusts the partitioning mix, ensuring consistent and optimal performance. Incorporated into this is a suite of implementation strategies like range-based, list-based, and hash-based partitioning. Each serves a unique purpose: from partitioning data based on predefined categories to evenly distributing data using hash values, ensuring there are no concentration hotspots.

B. Shard Selection and Ranking in HSSA

In distributed systems, data are spread across multiple shards, making global understanding a herculean task. HSSA alleviates this through the Central Sample Index (CSI). A representative set of documents from every shard is carefully sampled and indexed in the CSI. This curated index serves as a microcosm of the entire dataset, providing a bird's eye view of the information sprawl across all shards. By relying on this subset, HSSA can rapidly ascertain the potential relevance of vast amounts of data without trawling through every individual

shard. The efficiency of the CSI comes to the forefront when a query is dispatched. Rather than wandering aimlessly across shards, the query is first mapped against the CSI. Given that the CSI encapsulates samples from all shards, it is adept at offering an initial assessment of which shards are likely to be most relevant to the query in question. This preliminary direction ensures that search operations are both swift and pinpointed.

After the identification of potential shards, HSSA takes an additional step. It arranges these shards in a hierarchy, dictated by their estimated relevance to the posed query. By directing its computational prowess towards the highest-ranking shards first, HSSA ensures optimal resource utilization, leading to faster and more relevant search results. HSSA's architectural brilliance doesn't just reside in its native capabilities but also in its adaptability. While it carries a unique approach, it pays homage to time-tested shard selection algorithms like CORI and ReDDE. When scenarios demand it, HSSA can seamlessly amalgamate these methods into its operations. This synergy not only adds another layer of robustness to HSSA but also ensures its relevance across a spectrum of use-cases, making it a truly versatile tool in the domain of distributed data systems. CSI stands as a monumental pillar in the HSSA's framework, offering a streamlined approach to shard selection and relevance estimation.

C. Integrating Static and Dynamic Approaches

To combine the static and dynamic shard selection algorithms, we can use a linear combination of the algorithms. A weighted approach allows for prioritizing the strengths of each method based on their performance on the dataset.

$$HSSA(q,s) = \alpha \times (Lex(s,q) + Conn(s)) + \beta \times (CoRI(q,s) + ReDDE(s,q) + Rank - s(s,q)) \quad (1)$$

where α and β are weights representing the importance of the static and dynamic approaches, respectively.

The determination of α and β is vital. If, for example, static approaches (Lex-S and Conn-S) provide consistently reliable results in a particular dataset, α might be higher than β. Alternatively, if real-time adaptability and responsiveness are more crucial, then β might weigh more. To further fine-tune the hybrid approach:

$$HSSA(q,s) = \alpha1 \times Lex(s,q) + \alpha2 \times Conn(s) + \beta1 \times CoRI(q,s) + \beta2 \times ReDDE(s,q) + \beta3 \times Rank - s(s,q) \quad (2)$$

In this expanded equation, each algorithm has its weight (α1, α2, β1, β2, β3), allowing for granular control over the integration of each method into the hybrid approach. These weights can be optimized through several methods. In this study, empirical analysis was opted by running the HSSA with different weights on validation data and choosing the weights that offer the best results.

```
Algorithm 1: Proposed HSSA Algorithm
Require: List of shards retrieved from
distributed storage systems.
Ensure: Selected shards which are relevant
to the user query.
function hybrid_shard_selection(query):
    relevant_shards = [ ]
    static_selected_shards =
static_shard_selection(query)
    dynamic_selected_shards =
dynamic_shard_selection(query)
    combined_shards =
merge(static_selected_shards,
dynamic_selected_shards)
    relevant_shards =
rank_shards(combined_shards)
    return relevant_shards
function static_shard_selection(query):
    relevant_shards = [ ]
    for shard in shards:
    if shard.meets_static_criteria(query):
    relevant_shards.append(shard)
    return relevant_shards
function dynamic_shard_selection(query):
    relevant_shards = [ ]
    for shard in shards:
    if
shard.meets_dynamic_criteria(query):
    relevant_shards.append(shard)
    return relevant_shards
function merge(list1, list2):
// Combine two lists of shards while
removing duplicates
    merged_list = list1 + list2
    merged_list =
remove_duplicates(merged_list)
    return merged_list
function rank_shards(shards):
// Rank shards based on relevance score
    ranked_shards =
sort_shards_by_score(shards)
    return ranked_shards
```

IV. RESULTS AND DISCUSSION

The experiment was meticulously designed to ensure the comprehensive evaluation of HSSA against prevalent shard selection algorithms. The metrics chosen offered a holistic view of the algorithm's efficiency, effectiveness, and potential applicability in real-world scenarios.

A. Performance Analysis of HSSA Considering Latency

Figure 2 shows how different shard selection algorithms behave in terms of latency (ms) as the maximum number of shards (K) increases. The x-axis represents the maximum number of shards, whereas the y-axis denotes latency. It is evident that as K increases, latency generally tends to rise for all algorithms. This is expected as a larger value of K implies that more shards are considered, leading to higher computation time. However, HSSA consistently outperforms the other algorithms across all K values, achieving significantly lower latency. At K = 80, HSSA registers a latency of 21 ms, whereas the closest competitor, Rank-S, records a latency of 28 ms, underlining a clear efficiency gain. In contrast, CORI and Redde peak with latency values of 30 ms at 80 shards, with Lex-S and Conn-S escalating by 25 ms and 26 ms respectively.

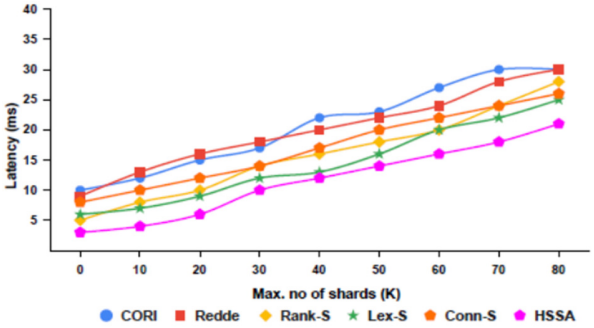

Fig. 2. Latency vs. maximun number of shards.

B. Average Document Score

Figure 3 shows the performance of various shard selection algorithms evaluated based on the average document score, a critical metric that signifies the average relevance of documents retrieved for specific queries. The proposed HSSA stands out distinctly, consistently outperforming its counterparts across all K values. For instance, at K = 80, HSSA achieves a score of 9.2, a testament to its efficacy. The underlying hybrid nature of HSSA, which seamlessly merges the merits of both static and dynamic shard selection methodologies, plays a pivotal role in its superior performance. Lex-S, while initiating on a modest note, exhibits a commendable growth, closely following HSSA with a score of 8.1 at K = 80. Rank-S, although beginning at 1.2, steadily ascends to 7.8 for K = 80. On the other hand, CORI and Redde display moderate growth trajectories, achieving scores of 6 and 6.5, respectively at K = 80. Conn-S, maintaining a consistent performance gradient, peaks at 7.5 for K = 80. To encapsulate, while each algorithm presents its unique advantages and growth trajectories, HSSA emerges as a clear frontrunner, underscoring its potential to set new standards in the domain of distributed processing systems.

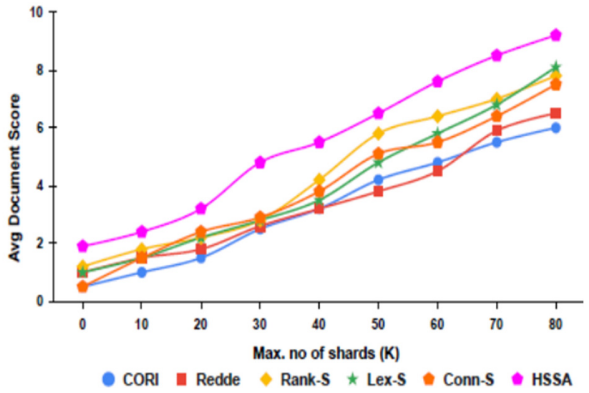

Fig. 3. Average document score vs. maximum number of shards.

C. Comparative Study Analysis

Figure 4 shows the Mean Average Precision (MAP), resource utilization cost and the latency for 60 queries on Gov2 dataset. We can observe that the MAP score of the proposed hybrid shard selection is high whereas the cost and latency are lower that those of the existing algorithms.

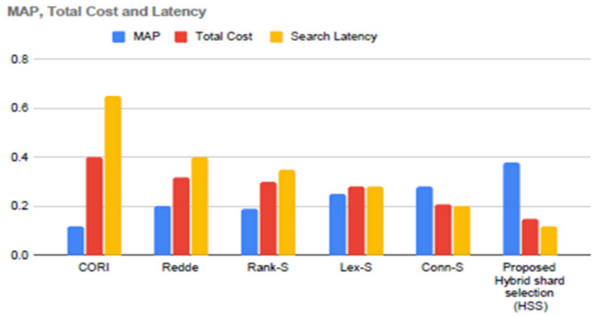

Fig. 4. MAP, total cost, and latency for the compared shard algorithms.

Table I shows the proposed algorithm's performance analysis compared to existing shard selection methods. The consistent precision at multiple cut-off points (P@10, P@30, and P@60) across all assessed K-values of HSSA substantiates its efficacy in discerning and retrieving relevant documents, even as the allowable shards burgeon. Its robust hybrid nature, amalgamating static and dynamic shard selection approaches, underpins its superior performance by ensuring relevance and minimized computational overhead.

TABLE I. PERFORMANCE ANALYSIS OF SHARD SELECTION ALGORITHMS CONSIDERING PRECISION, COST, AND LATENCY PARAMETERS

Shard selection algorithm	No. of shards	Search cost					
		P@30	P@60	NDCG@100	MAP	C_{Total}	$C_{Latency}$
Exhaustive	50	0.53	0.47	0.41	0.27	2.87	2.89
	100	0.53	0.47	0.41	0.27	2.87	0.96
	150	0.53	0.47	0.41	0.27	2.87	0.58
	200	0.53	0.47	0.41	0.27	2.87	0.51
CORI	50	0.28	0.3	0.24	0.14	0.12	0.12
	100	0.36	0.38	0.32	0.21	0.32	0.21
	150	0.42	0.42	0.37	0.25	0.41	0.24
	200	0.51	0.48	0.4	0.26	0.48	0.26
ReDDE	50	0.38	0.31	0.25	0.13	0.12	0.12
	100	0.47	0.44	0.36	0.23	0.36	0.18
	150	0.52	0.46	0.39	0.25	0.47	0.21
	200	0.56	0.51	0.43	0.28	0.52	0.28
Rank-S	50	0.34	0.3	0.2	0.16	0.14	0.12
	100	0.45	0.34	0.29	0.28	0.38	0.19
	150	0.49	0.45	0.34	0.29	0.48	0.21
	200	0.51	0.48	0.41	0.4	0.52	0.28
Modified Rank-S	50	0.34	0.3	0.2	0.16	0.14	0.16
	100	0.45	0.34	0.29	0.28	0.25	0.14
	150	0.49	0.45	0.34	0.29	0.3	0.18
	200	0.51	0.48	0.41	0.4	0.32	0.21
Proposed HSSA	50	0.3	0.31	0.3	0.28	0.16	0.15
	100	0.28	0.3	0.28	0.27	0.2	0.12
	150	0.27	0.28	0.26	0.25	0.21	0.15
	200	0.23	0.26	0.25	0.24	0.24	0.16

V. CONCLUSION

This paper presents a new approach to the distributed data processing problem: the Hybrid Shard Selection Algorithm (HSSA). The existing shard selection algorithms focus on search efficiency without significantly reducing search effectiveness. They may suffer from high query latency and load imbalance when the size of the database is enormous. The proposed hybrid shard selection approach partitions the

documents into topic-wise shards and uses cluster-skipping indexes inside each of the shards during the query processing operation. Only those shards that have relevant documents matching the user query are searched. This reduces the latency and enhances the search effectiveness by reducing the resource utilization cost. We tested the proposed algorithm on the Gov2 dataset partitioned into topic-based shards. The proposed HSSA algorithm search accuracy and effectiveness are significantly better than those of the baseline existing shard selection algorithms. At the same time, the search cost of the proposed algorithm is lower with optimized resource utilization. It reduced the search cost by 20% for the Gov2 dataset. The proposed algorithm supports query-specific minimal shard cut-off estimation, which can be used with different shard ranking algorithms to satisfy early precision or high recall goals of information retrieval applications.

REFERENCES

[1] N. Venkateswaran and S. Changder, "Simplified data partitioning in a consistent hashing based sharding implementation," in *TENCON 2017 - 2017 IEEE Region 10 Conference*, Penang, Malaysia, Aug. 2017, pp. 895–900, https://doi.org/10.1109/TENCON.2017.8227985.

[2] A. Kulkarni, A. S. Tigelaar, D. Hiemstra, and J. Callan, "Shard ranking and cutoff estimation for topically partitioned collections," in *Proceedings of the 21st ACM international conference on Information and knowledge management*, New York, NY, USA, Jul. 2012, pp. 555–564, https://doi.org/10.1145/2396761.2396833.

[3] J. Kamal, M. Murshed, and R. Buyya, "Workload-aware incremental repartitioning of shared-nothing distributed databases for scalable OLTP applications," *Future Generation Computer Systems*, vol. 56, pp. 421–435, Mar. 2016, https://doi.org/10.1016/j.future.2015.09.024.

[4] H. R. Mohammad, K. Xu, J. Callan, and J. S. Culpepper, "Dynamic Shard Cutoff Prediction for Selective Search," in *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, New York, NY, USA, Mar. 2018, pp. 85–94, https://doi.org/10.1145/3209978.3210005.

[5] P. M. Dhulavvagol, V. H. Bhajantri, and S. G. Totad, "Performance Analysis of Distributed Processing System using Shard Selection Techniques on Elasticsearch," *Procedia Computer Science*, vol. 167, pp. 1626–1635, Jan. 2020, https://doi.org/10.1016/j.procs.2020.03.373.

[6] Z. Dai, C. Xiong, and J. Callan, "Query-Biased Partitioning for Selective Search," in *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management*, New York, NY, USA, Jul. 2016, pp. 1119–1128, https://doi.org/10.1145/2983323.2983706.

[7] A. Kulkarni and J. Callan, "Selective Search: Efficient and Effective Search of Large Textual Collections," *ACM Transactions on Information Systems*, vol. 33, no. 4, pp. 17:1-17:33, Dec. 2015, https://doi.org/10.1145/2738035.

[8] P. M. Dhulavvagol, S. G. Totad, and S. Sourabh, "Performance Analysis of Job Scheduling Algorithms on Hadoop Multi-cluster Environment," in *Emerging Research in Electronics, Computer Science and Technology*, Singapore, 2019, pp. 457–470, https://doi.org/10.1007/978-981-13-5802-9_42.

[9] N. C. Kundur, B. C. Anil, P. M. Dhulavvagol, R. Ganiger, and B. Ramadoss, "Pneumonia Detection in Chest X-Rays using Transfer Learning and TPUs," *Engineering, Technology & Applied Science Research*, vol. 13, no. 5, pp. 11878–11883, Oct. 2023, https://doi.org/10.48084/etasr.6335.

[10] E. Rodrigues and R. Morla, "Run Time Prediction for Big Data Iterative ML Algorithms: a KMeans case study," Oct. 2017.

[11] M. Ali, N. Q. Soomro, H. Ali, A. Awan, and M. Kirmani, "Distributed File Sharing and Retrieval Model for Cloud Virtual Environment," *Engineering, Technology & Applied Science Research*, vol. 9, no. 2, pp. 4062–4065, Apr. 2019, https://doi.org/10.48084/etasr.2662.

[12] N. Jayakumar and A. M. Kulkarni, "A Simple Measuring Model for Evaluating the Performance of Small Block Size Accesses in Lustre File System," *Engineering, Technology & Applied Science Research*, vol. 7, no. 6, pp. 2313–2318, Dec. 2017, https://doi.org/10.48084/etasr.1557.