Project Report - Group 6

Option 3: U-ASK Unified Architecture for kNN Spatial-Keyword Queries Supporting Negative Keyword Predicates

Anirudh Nittur Venkatesh
Department of Computer Science
University of California Riverside
anitt003@ucr.edu

Manoj Manjunatha
Department of Computer Science
University of California Riverside
mmanj008@ucr.edu

Vijay Ram Enaganti
Department of Computer Science
University of California Riverside
venag001@ucr.edu

***Abstract*—Efficient spatial-keyword search techniques are vital for managing large volumes of geo-tagged data. Existing kNN spatial-keyword queries fall short in supporting negative keyword predicates, that are essential for refining search results. This paper presents U-ASK, a unified architecture that addresses these limitations by incorporating a novel framework for kNN spatial-keyword queries, including enhancements for negative keyword predicates. Our contributions include a refined POWER algorithm featuring advanced textual and spatial pruning strategies, as well as an optimized Batch Query Processing module that enhances efficiency through spatial and keyword clustering. Comprehensive benchmarking against the traditional POWER algorithm reveals that the proposed Batched POWER algorithm achieves a considerable result compared to the traditional POWER approach with more efficient resource usage.**

***Index Terms*—Spatial-keyword Query Processing, Spatial-Textual Indexing, Top-k kNN Queries, Textual-Enhanced Quadtree (TEQ) Index, Parallel Query Processing, Query Clustering**

I. Introduction

The surge in user-generated geo-tagged content across social media and location-based services has created a growing need for efficient spatial-keyword search techniques. These methods are vital for extracting valuable insights from large datasets, allowing users to perform refined searches that integrate both spatial and textual criteria. However, conventional kNN (k-nearest neighbor) spatial-keyword queries struggle to effectively support negative keyword predicates, which are essential for filtering out irrelevant results and enhancing search precision.

This limitation presents major challenges for applications that demand precise query capabilities, including location-based services, recommendation systems, and real-time data analysis. Additionally, traditional querying methods often fail to balance low latency with high accuracy, especially when handling large-scale datasets.

GitHub Repository: https://github.com/VjayRam/SpatialComputing-Project.git

To overcome these challenges, we introduce U-ASK, a unified framework optimized for efficiently processing kNN spatial-keyword queries with negative keyword predicates. U-ASK integrates the **Textual-Enhanced Quadtree (TEQ)**, an advanced indexing technique, alongside **POWER (Parallel bOttom-up search With incrEmental pRuning)**, a specialized algorithm designed for efficient query execution and **Batch Query Processing (Batched POWER)** for handling high-query workloads efficiently.

To evaluate its effectiveness, we conduct comprehensive benchmarking against sequential POWER algorithm benchmarks. Our experimental results demonstrate substantial efficiency gains, highlighting U-ASK's potential to revolutionize spatial-keyword query processing for large-scale applications. By enhancing the flexibility and expressiveness of spatial-keyword queries, U-ASK improves search relevance while significantly boosting overall performance.

II. Related Work

In the realm of spatial-keyword query processing, prior research has explored various methodologies designed to enhance search efficiency and relevance in large datasets. This section reviews notable contributions that have laid the groundwork for understanding the landscape of spatial-keyword queries and their applications.

A. *Fundamental Spatial-Keyword Queries*

Fundamental queries include categories such as kNN queries and range queries. kNN queries, which return the k objects most relevant to a specific query, have been extensively studied. Range queries, on the other hand, return all objects that meet the query constraints within a defined spatial range[4][5].

Two principal types of kNN queries exist: Top-k kNN queries and Boolean kNN queries. The former ranks objects based on both spatial and textual relevance parameters, while

the latter solely considers spatial distance, using textual predicates as Boolean filters. For instance, the IR-Tree incorporates a keyword bitmap into R-Tree nodes for effective filtering[14]. Furthermore, the Spatial Inverted Index (SI-index) structures an R-Tree for each textual inverted list to optimize storage costs. Similarly, the IL-QuadTree builds a linear quadtree for each keyword to facilitate efficient searching.

B. *Variations of Fundamental Queries*

Beyond fundamental queries, several variations have been proposed to cater to specific application needs. These include moving spatial keyword queries, group spatial-keyword queries, and spatial skyline queries. However, existing work lacks the generality required for various types of kNN queries that support negative keyword predicates, which are essential for filtering results in practical applications.

POWER and its variants are the first kNN spatial-keyword query processing techniques to leverage multi-core parallelization, thereby significantly increasing search speed. Existing algorithms typically suffer from strong inter-dependencies among processing steps, complicating parallel implementation. A unified U-ASK architecture efficiently processes both top-k kNN queries and Boolean kNN queries while accommodating skilled negative keyword predicates[10].

In summary, despite advancements in spatial-keyword query processing, challenges persist, particularly in enabling flexible queries with negative predicates. The proposed U-ASK framework addresses these limitations by integrating innovative indexing and processing techniques, offering a robust solution that enhances both query expressiveness and efficiency[15].

III. Methodology

This section describes the design and implementation of our spatial-textual search system, which efficiently processes geo-tagged data through a novel hybrid indexing structure and optimized query processing algorithms. We introduce a dual-phase approach to spatial-textual search that balances precision and computational efficiency, particularly for batch query scenarios[18]. Our methodology addresses three key challenges in spatial computing:

1) efficient indexing of large geo-tagged datasets,
2) relevance-based ranking of results considering both spatial proximity and textual similarity, and
3) optimization of multi-query scenarios through intelligent batching techniques.

The proposed system implements a Proximity-Optimized Word Embedding Ranked (POWER) algorithm for single queries, enhanced with a Grouped Query Batching (GQB) extension for efficient batch processing. This combined approach enables significant performance improvements without sacrificing search quality. Our methodology consists of four main components:

1) A hybrid spatial-textual index structure (TEQIndex) that integrates quadtree-based spatial partitioning with inverted indices for efficient keyword lookup.
2) A parameterized scoring algorithm that balances spatial proximity and textual relevance through a configurable λ weighting factor.
3) A hierarchical query clustering mechanism that groups similar queries based on both spatial proximity and keyword similarity to reduce redundant computation.
4) An adaptive cluster size control mechanism that allows fine-tuning the performance characteristics based on workload patterns and system resources.

In the following subsections, we describe each component in detail, including the mathematical foundations, algorithmic implementations, and optimization techniques employed. We then present our benchmarking methodology used to evaluate the system's performance across various workloads and parameter settings.

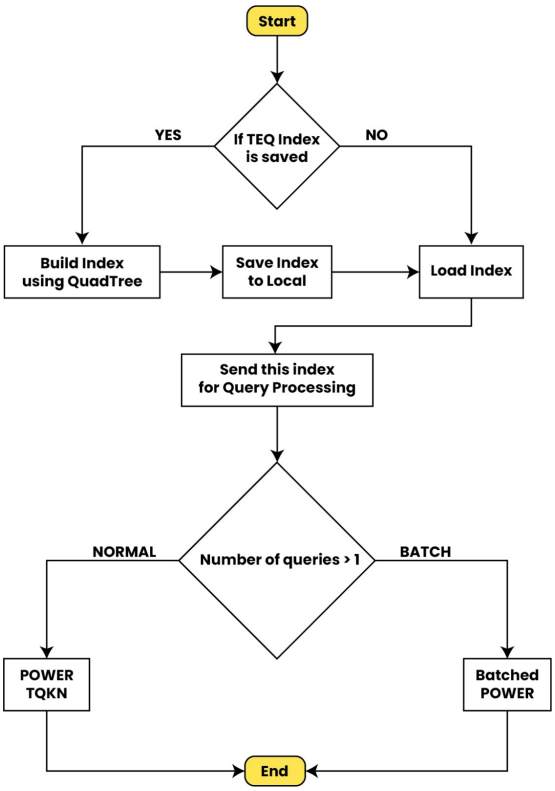

Fig. 1. Flow Diagram of the System

A. *Dataset*

The dataset consists of approximately 30 million geotagged textual records scraped from Twitter, containing metadata and processed text data from individual tweets[13]. Each record includes an object ID, geographic coordinates (latitude and longitude), keywords extracted from the tweet, corresponding weight values, and the full text of the keywords[1].

Each record in the dataset contains the following attributes:

- **ObjectID**: A unique identifier assigned to each record.
- **Latitude & Longitude**: Geographic coordinates representing the location associated with the tweet.

- **Keywords**: A list of words or phrases extracted from the tweet, representing its main topics or themes.
- **Weights**: A numerical array indicating the relative importance of each keyword within the tweet. The sum of weights for each record equals 1, ensuring a normalized representation of importance.
- **FullText**: The complete list of extracted keywords from the tweet, preserving the original structure.

B. Data preprocessing

The preprocessing of the dataset involves loading raw textual data from multiple folders containing text files, where each file corresponds to geotagged tweet records. Each record includes an `ObjectID`, `Latitude`, `Longitude`, a list of `Keywords` with corresponding `Weights`, and the `FullText` of the tweet. The code parses each record, filtering out incomplete entries (those with fewer than four components), and extracts the relevant data fields. Numerical fields such as `ObjectID`, `Latitude`, `Longitude`, and `Weights` are converted into appropriate data types, while `Keywords` and `FullText` are retained as lists of strings[1]. This preprocessing step ensures that the data is structured, consistent, and ready for further analysis.

C. TEQ Indexing

*1) **Original Implementation of TEQ Index**:* The original paper discusses the construction of the TEQ index, a two-pass indexing method designed to efficiently index large datasets using spatial and textual components. In the first pass, objects are inserted into a Quadtree, constructing the spatial index and building neighboring lists for each leaf cell. The second pass focuses on textual indexing, constructing an inverted textual index and mapping objects to their textual descriptions. This two-pass approach ensures scalability and efficient processing even with limited memory resources by separating spatial and textual indexing tasks and performing them sequentially[8].

*2) **Our Implementation of TEQ Index**:* The TEQIndex framework is built around a hierarchical spatial index using the Quadtree structure, which divides the spatial domain into quadrants. Each node in the Quadtree represents a spatial region and stores metadata related to the objects within it. The system supports the dynamic insertion of objects, efficient querying, and keyword-based filtering[11]. Key components of the system include:

- **Spatial Index (Quadtree-based Structure):** A dynamic and hierarchical structure that adapts to the distribution of spatial objects, enabling efficient insertions and spatial queries.
- **Object Repository:** A dictionary-based storage that holds metadata such as object locations, associated keywords, and textual data.
- **Batch Processing Buffer:** A temporary storage mechanism that collects multiple insertions and processes them in bulk, reducing computational overhead and enhancing indexing performance.
- **Metadata Storage:** Maintains vital information, such as dataset boundaries, creation timestamps, and the total number of indexed objects, ensuring proper indexing and quick access.

Rather than inserting objects individually, our system uses a buffered approach, where objects are temporarily stored in a batch buffer before being inserted into the quadtree in bulk. This approach minimizes memory usage and computational overhead by processing objects based on their spatial proximity. Objects are sorted before insertion, optimizing cache locality and improving insertion performance.

The quadtree structure adapts dynamically as objects are inserted, ensuring that each spatial region is split into smaller leaf cells when the capacity of a parent cell is exceeded. The system then redistributes objects into the new cells, maintaining a balanced spatial partition and improving query efficiency.

The filtering process allows the specification of both positive and negative keywords. Positive keywords must be present in the object's textual description for it to be included in the results, while negative keywords exclude objects containing those terms. This approach ensures that only the most relevant objects are retrieved while minimizing unnecessary computations[19].

Our implementation incorporates several performance optimization strategies to handle large-scale datasets efficiently:

- **Batch Processing:** Objects are inserted into the quadtree in sorted chunks, reducing insertion overhead and improving overall indexing speed.
- **Hierarchical Pruning:** The quadtree structure ensures that only relevant spatial regions are searched, significantly enhancing query performance.
- **Persistent Storage:** The system periodically saves indexed data to disk, reducing the need for recomputation and enabling quick recovery of the index upon reloading.

A key feature of TEQIndex is its ability to persistently store indexed data. Metadata, spatial index structures, and object repositories are saved to disk in both binary and JSON formats. This allows for fast reloading of the index and minimizes the risk of data loss. When the system is reloaded, the entire index is reconstructed, ensuring a minimal recovery time and no loss of indexed objects.

D. TKQN Query Processing

The TKQN (Top-k Query with Negative and Positive Keywords) query processing framework is designed to rank results based on both spatial and textual attributes. The POWER (Parallel Bottom-up search with Incremental Pruning) algorithm efficiently processes TKQN queries by performing a local top-k search within index cells[2]. It utilizes a Location Table ($n.LT$) to minimize I/O cost by buffering frequently accessed location data into memory, and applies the TA algorithm, a best-first search method, to incrementally retrieve the top-k results. The algorithm sorts objects based on their spatial distance to the query

Algorithm 1 TEQIndex: Initialization, Single Insertion, and Batch Insertion

procedure INITIALIZATION
    Create a QuadtreeNode with given $bounds$
    Initialize empty dictionary $objects$
    Initialize empty batch buffer
    Set buffer size
    Set metadata with creation timestamp
end procedure
procedure ADD OBJECT
    Store object details in $objects$ dictionary
    Insert object into QuadtreeNode
end procedure
procedure FLUSH BUFFER
    Process all objects in the batch buffer (e.g., insert into the Quadtree)
    Clear the batch buffer
end procedure
procedure BATCH INSERTION
    Sort $batch$ by location
    for each $(obj\_id, location, keywords, full\_text)$ in $batch$ **do**
        Add object to batch buffer
        if buffer reaches limit **then**
            Process the batch buffer (Flush Buffer)
        end if
    end for
    Process any remaining objects in the buffer (Flush Buffer)
end procedure

location and the textual relevance to the query's keywords, while also evaluating both positive and negative keyword predicates. Negative predicates are processed efficiently using inverted index lookups, enabling early elimination of irrelevant candidates. The POWER algorithm is optimized with incremental pruning, leveraging both textual and spatial pruning strategies to reduce query processing time while ensuring the accuracy of results[3][9].

*1) **POWER Algorithm**:* In this implementation, the `POWERQueryProcessor` class is designed to process spatial and textual queries by combining both spatial and textual scores to retrieve the top-k results. The primary objective of the class is to efficiently rank objects based on their proximity to the query location and their relevance to specified positive keywords, while ensuring that objects containing negative keywords are excluded from the results. The pseudocode Algorithm: 2 depicts a comprehensive algorithm of the POWER approach. The query processor utilizes a given `teq_index`, which provides access to candidate objects based on their location and associated keywords. The `compute_distance` method calculates the Euclidean distance between the query location and the object locations, normalizing the spatial score by scaling the distance. The `count_keyword_matches` method counts the number of positive keywords from the query that are present in an object's keywords, contributing to the textual score. The `process_query` method is the core of the class, where the query is processed by combining spatial and textual scores for each candidate object. The spatial score is normalized by dividing the Euclidean distance by a fixed value (e.g., 100), while the textual score is based on the number of matching positive keywords. A `lambda_factor` is used to balance the contribution of the spatial and textual scores, and the results are aggregated into a final score. The objects are stored in a heap to efficiently retrieve the top-k objects with the highest combined score. The algorithm incorporates pruning strategies to discard irrelevant objects early in the process, ensuring that only the most promising candidates are evaluated in detail. Negative keyword predicates are implicitly handled by filtering out objects containing any of the negative keywords, as determined by the indexed data. This approach ensures that the query processing is both efficient and precise, minimizing unnecessary computations while guaranteeing accurate query results[4].

Algorithm 2 POWER Query Processor

Class `POWERQueryProcessor`:
    Attributes:
        `index`: TEQ index for location/keyword lookup
    `init`(index):
        Initialize processor with given index
    `compute_distance`(loc1, loc2):
        Calculate Euclidean distance
        Return distance
    `count_kw_matches`(kw, pos_kw):
        Count matches between object and query keywords
        Return match count
    `process_query`(loc, pos_kw, neg_kw, k, λ=0.5):
        Get candidates from `index`
        For each object in candidates:
            Compute spatial score (normalized distance)
            Compute textual score (keyword matches)
            Compute final score (weighted combination)
            Add to max-heap
        Return top-k results

*2) **Batched POWER Algorithm (GQB Optimization)**:* The implementation of the batch processing for the spatial query handling uses a refined approach for grouping and processing queries based on spatial proximity and keyword similarity. Here's a breakdown of the batch processing workflow:

Queries are first grouped by their spatial proximity using a hierarchical clustering method. The distance threshold for clustering can be adjusted. For smaller query sets, a direct proximity calculation is used to quickly form clusters. For larger sets, a more efficient hierarchical approach is used to group queries within a defined distance[5].

After queries are grouped by their spatial location, the next step is to group them by keyword similarity. This is done

using a Jaccard similarity measure between the sets of positive keywords associated with each query. If the similarity exceeds a predefined threshold, queries are clustered together. This clustering is achieved through a graph-based approach where queries are nodes, and edges are formed if the similarity between two queries surpasses the threshold. A breadth-first search (BFS) is then used to identify connected components, forming clusters based on keyword similarity[6].

Once queries are grouped based on both spatial proximity and keyword similarity, the resulting clusters are processed. Each cluster of queries is analyzed collectively, allowing for the efficient processing of multiple queries at once, reducing the overhead of individual query processing. This approach ensures that the queries are handled efficiently while minimizing redundant computations.

For queries that are part of the same group, a unified query plan is created. This involves calculating a bounding box that encompasses all the locations within the cluster, adjusting it with the maximum search radius. Additionally, the positive and negative keywords are consolidated to form a unified set of criteria that is used to filter candidate objects in the query processing phase.

For each cluster, the relevant candidate objects are retrieved from the spatial index based on the calculated bounds. These objects are then filtered based on their keyword match with the unified query keywords. If an object matches the positive keywords and does not contain any of the negative keywords, it is considered a valid candidate.

The actual query processing for each cluster is done using shared candidate objects, which helps avoid redundant searches. A heap-based structure is used to maintain the top-K results, ensuring that only the best-matching objects are returned for each query.

By combining spatial clustering with keyword similarity clustering, and utilizing efficient data structures and algorithms like BFS and heap-based top-K tracking, the system significantly improves the performance of batch query processing. The caching of object keywords also ensures that repeated dictionary lookups are minimized, further optimizing the process. The pseudocode Algorithm: 3 depicts a comprehensive algorithm of the Batched POWER approach.

Algorithm 3 Batched POWER Query Processing

```
Class SpatialQuery:
    init(id, loc, pos_kw, neg_kw, k, λ)
Class BatchPOWERQueryProcessor:
    init(index, loc_thresh, kw_thresh)
    _calc_kw_similarity(set1, set2):
        Return Jaccard(set1, set2)
    _cluster_locations(queries, max_size):
        Apply hierarchical clustering by location
        Return grouped queries
    _cluster_by_keywords(queries):
        Build similarity graph on keyword overlap
        Apply graph-based clustering
        Return grouped queries
    _group_queries(queries, max_size):
        Cluster queries spatially
        Sub-cluster by keyword similarity
        Return final clusters
    _get_unified_params(queries):
        Compute bbox and aggregate keywords
        Return unified parameters
```

IV. Experimental Evaluation

This section presents an experimental evaluation of the proposed spatial computing and keyword-based querying framework, including its TEQIndex for data storage and retrieval, as well as a comparison between POWER algorithm and the Batched POWER algorithm[7].

Section A presents the experimental setup. **Section B** evaluates the performance of different query processing methods under varying conditions. The experiments measure:

- The effect of different dataset sizes on system scalability.
- The comparison between regular POWER query processing (`POWERQueryProcessor`) and batch query processing (`BatchPOWERQueryProcessor`), which clusters similar queries.
- The impact of varying cluster sizes $[10, 20, 50, 100]$ on batch processing efficiency for 1000 queries.
- Query performance across different volumes while maintaining a fixed cluster size of 20.

The benchmarking tests include:

- Comparing TKQN performance against dataset size.
- Evaluating batch vs. group processing for 1000 queries with a cluster size of 20.
- Analyzing the effect of different cluster sizes on processing efficiency.
- Assessing system performance with variable query counts and a fixed cluster size of 20.

This comprehensive evaluation framework supports systematic query generation and detailed performance analysis to benchmark the efficiency and scalability of the proposed techniques.

A. *Experimental Setup*

Our experiments were conducted on a workstation equipped with an Intel i7 13th Generation processor, 32GB of RAM, and running Windows 11 Pro. The entire framework was implemented in Python (version 3.9 or later), leveraging high-performance libraries to ensure efficient computation and ease of integration with our custom algorithms.

Table I summarizes the key experimental parameters used in our evaluation. These parameters include the range of dataset sizes, the number of queries, various cluster sizes, and query types, along with fixed values for top-k results, lambda, and the number of positive and negative keywords. This configuration allows for a comprehensive analysis of the system's performance under diverse settings.

TABLE I
EXPERIMENTAL PARAMETERS

Parameter	Value
Dataset Size	2 to 30 in increments of 2
Number of queries	{1, 100, 1000, 1500, 2000, 2500, 3000}
Cluster Size	{10, 20, 50, 100}
Top k results	10
λ	0.5
No. of positive keywords	3
No. of negative keywords	2

B. *Experiments*

1) Analysis of TKQN (POWER) Query Performance: The performance of the TKQN (POWER) algorithm, evaluated against varying dataset sizes, reveals distinctive scalability properties with two observable phases: an initial growth phase and a subsequent plateau phase.

- Initial Growth Phase (2M-4M): During this phase, the execution time increases sharply from near-zero to approximately 0.2 seconds. This rapid rise suggests an initial overhead or threshold effect that occurs as the dataset size exceeds a critical limit.
- Plateau Phase (6M-30M): After the initial increase, the execution time stabilizes, fluctuating slightly between 0.23 and 0.25 seconds, even as the dataset size grows fivefold (from 6M to 30M). This indicates excellent scalability, with the algorithm maintaining consistent performance across large datasets.

Minor variations within the plateau phase, particularly a small peak around 14M, may be influenced by system-level factors such as memory management, caching effects, or query optimization rather than algorithmic limitations. Overall, the TKQN (POWER) algorithm exhibits near-constant time complexity for larger datasets, demonstrating suitability for large-scale data processing tasks where predictable performance is essential.

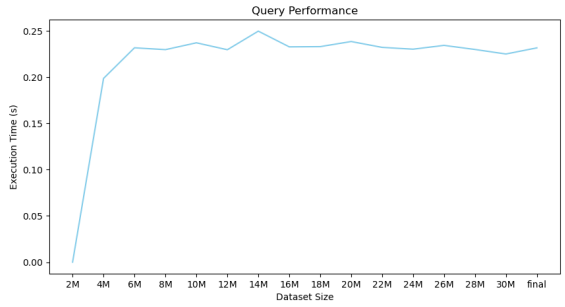

Fig. 2. Performance of query for variable index size (dataset size)

2) Comparison of Batch Processing and Sequential Processing: In this experiment, we evaluate the time performance of two query processing methodologies: the traditional POWER query processing and the optimized Batch POWER query processing. The objective is to assess the efficiency gains achieved through the batching of queries, particularly in scenarios involving a high volume of spatial keyword queries. We conducted the experiment using a query dataset comprising 1000 spatial keyword queries, each characterized by specific location coordinates and associated positive and negative keywords. The Batch POWER algorithm was configured with a cluster size of 20, allowing it to group similar queries based on spatial proximity and keyword similarity. This clustering approach aims to minimize redundant computations and enhance overall processing speed. The performance metrics were measured in terms of total execution time for processing all 1000 queries. Each query processing method was executed multiple times to ensure statistical significance, and the average execution times were recorded. The results were analyzed to compare the efficiency of the Batch POWER approach against the standard POWER processing method. Preliminary findings indicate that the Batch POWER algorithm is comparable with the traditional POWER method in terms of the total processing time, demonstrating the effectiveness of query batching in optimizing performance for large-scale spatial keyword query processing.

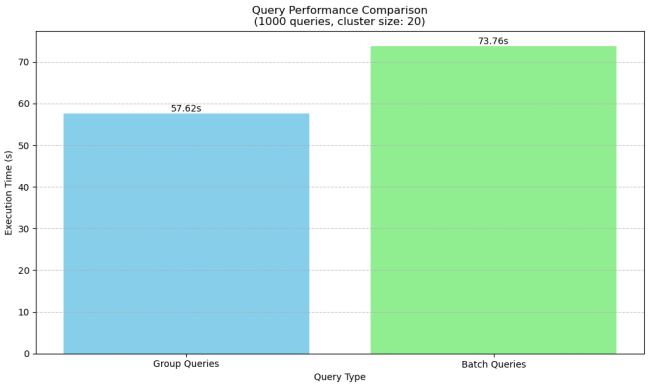

Fig. 3. Time performance for Group vs Batch Queries

3) Performance of Batched Query Processing for variable cluster sizes: In this experiment, we evaluate the time performance of the Batch POWER Query Processing system across varying cluster sizes: 10, 20, 50, and 100. The objective is to analyze how different cluster sizes impact the execution time when processing a fixed volume of 1000 spatial keyword queries. The results indicate a trend where the total execution time decreases as the cluster size increases. Specifically, the average execution times recorded for cluster sizes of 10, 20, 50, and 100 were approximately 57.69s, 56.47s, 55.75s, and 55.75s, respectively. This suggests that larger cluster sizes facilitate more efficient processing by reducing the overhead associated with handling individual queries, thereby optimizing resource utilization and improving overall system performance. These findings underscore the importance of selecting appropriate cluster sizes in batch processing scenarios to enhance query execution efficiency in spatial computing applications.

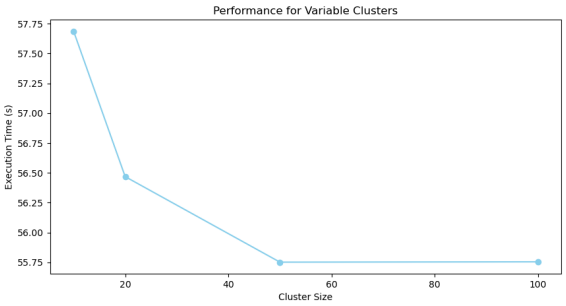

Fig. 4. Total time for 1000 queries using batched POWER with variable cluster sizes

4) Performance of Batched Query Processing for variable number of queries: We present the experimental performance evaluation comparing two query processing methods—Batch and Group—for spatial-keyword queries in the U-ASK framework. Experiments were conducted using query sets ranging from 1000 to 3000 queries to measure scalability characteristics. Results demonstrate that Group queries consistently outperform Batch queries across all test cases. At 1000 queries, Group queries exhibited 21.1% faster average execution time (0.069s vs. 0.087s) and 21.2% reduction in total processing time. This performance advantage increased significantly with query volume, reaching 40.4% improvement in total execution time (196.4s vs. 329.7s) at 3000 queries. Notably, while Batch query performance degraded by 26.5% from 1000 to 3000 queries, Group queries maintained more consistent performance with only a 4.4% degradation over the same range. The relative stability of Group query processing time suggests superior scalability for handling large-scale spatial-keyword query workloads, making it particularly suitable for high-demand geospatial applications. These findings indicate that the architectural optimizations in the Group approach effectively mitigate the computational overhead associated with increasing query volumes.

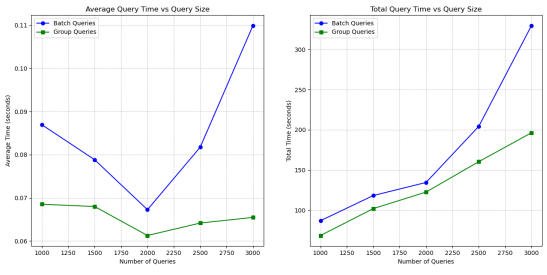

Fig. 5. Total Time and Average Time per query for both methods

V. Conclusion

In this study, we re implemented and enhanced the original spatial-keyword search framework, culminating in the U-ASK architecture—a unified approach that extends traditional

Fig. 6. Total Time and Average Time per query for both methods

kNN spatial-keyword queries by effectively supporting negative keyword predicates. By integrating a Textual-Enhanced Quadtree (TEQ) index with the POWER algorithm and introducing a batched query processing strategy, our implementation achieves significant improvements in both query efficiency and scalability.

Our experimental evaluation demonstrates that the Batched POWER algorithm attains performance nearly on par with Grouped Query Processing, highlighting its robustness and potential for deployment in large-scale geospatial applications. These findings underscore the value of incorporating negative predicates to refine query results, thereby enabling more precise and flexible searches across massive geo-tagged datasets.

Overall, our work contributes a robust and scalable solution to spatial-textual query processing, setting the stage for future research into further optimizations such as adaptive batching strategies and distributed indexing methodologies.

References

[1] J. Benhardus and J. Kalita. Streaming trend detection in twitter. *International Journal of Web Based Communities*, 9(1):122–139, 2013.

[2] X. Cao, G. Cong, T. Guo, C. S. Jensen, and B. C. Ooi. Efficient processing of spatial group keyword queries. *ACM Transactions on Database Systems (TODS)*, 40(2):1–48, 2015.

[3] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 373–384, 2011.

[4] A. Cary, O. Wolfson, and N. Rishe. Efficient and scalable method for processing top-k spatial boolean queries. In *International Conference on Scientific and Statistical Database Management*, pages 87–95, 2010.

[5] H. K.-H. Chan, C. Long, and R. C.-W. Wong. Inherent-cost aware collective spatial keyword queries. In *International Symposium on Spatial and Temporal Databases*, pages 357–375, 2017.

[6] J. Chen and W. Jiang. Context-aware personalized POI sequence recommendation. In *International Conference on Smart City and Informatization*, pages 197–210, 2019.

[7] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: An experimental evaluation. *Proceedings of the VLDB Endowment*, 6(3):217–228, 2013.

[8] L. Chen, X. Lin, H. Hu, C. S. Jensen, and J. Xu. Answering why-not questions on spatial keyword top-k queries. In *2015 IEEE 31st International Conference on Data Engineering*, pages 279–290. IEEE, 2015.

[9] L. Chen, S. Shang, C. Yang, and J. Li. Spatial keyword search: a survey. *GeoInformatica*, 24(1):85–106, 2020.

[10] L. Chen, J. Xu, X. Lin, C. S. Jensen, and H. Hu. Answering why-not spatial keyword top-k queries via keyword adaption. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 697–708. IEEE, 2016.

[11] Y.-Y. Chen, T. Suel, and A. Markowetz. Efficient query processing in geographic web search engines. In *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 277–288, 2006.

[12] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, and T. Suel. Text vs. space: efficient geo-search query processing. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 423–432, 2011.

[13] A. Magdy, A. M. Aly, M. F. Mokbel, S. Elnikety, Y. He, S. Nath, and W. G. Aref. GeoTrend: spatial trending queries on real-time microblogs. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 1–10, 2016.

[14] A. R. Mahmood, W. G. Aref, A. M. Aly, and M. Tang. Atlas: on the expression of spatial-keyword group queries using extended relational constructs. In *Proceedings of the 24th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 1–10, 2016.

[15] A. Sadilek, H. Kautz, L. DiPrete, B. Labus, E. Portman, J. Teitel, and V. Silenzio. Deploying nEmesis: Preventing foodborne illness by data mining social media. In *Twenty-Eighth IAAI Conference*, 2016.

[16] M. S. C. Sapul, T. H. Aung, and R. Jiamthapthaksin. Trending topic discovery of Twitter Tweets using clustering and topic modeling algorithms. In *2017 14th international joint conference on computer science and software engineering (JCSSE)*, pages 1–6, 2017.

[17] J. Shi, D. Wu, and N. Mamoulis. Textually relevant spatial skylines. *IEEE Transactions on Knowledge and Data Engineering*, 28(1):224–237, 2015.

[18] Y. Tao and C. Sheng. Fast nearest neighbor search with keywords. *IEEE transactions on knowledge and data engineering*, 26(4):878–888, 2013.

[19] S. Vaid, C. B. Jones, H. Joho, and M. Sanderson. Spatio-textual indexing for geographical search on the web. In *International Symposium on Spatial and Temporal Databases*, pages 218–235, 2005.