Load-balanced Range Query Workload Partitioning for Compressed Spatial Hierarchical Bitmap (cSHB) Indexes

by

Ashish Gadkari

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved November 2018 by the
Graduate Supervisory Committee:

Kasim Selçuk Candan, Chair
Hasan Davulcu
Maria Luisa Sapino

ARIZONA STATE UNIVERSITY

December 2018

ABSTRACT

The spatial databases are used to store geometric objects such as points, lines, polygons. Querying such complex spatial objects becomes a challenging task. Index structures are used to improve the lookup performance of the stored objects in the databases, but traditional index structures cannot perform well in case of spatial databases. A significant amount of research is made to ingest, index and query the spatial objects based on different types of spatial queries, such as range, nearest neighbor, and join queries. Compressed Spatial Bitmap Index *(cSHB)* structure is one such example of indexing and querying approach that supports spatial range query workloads (set of queries). cSHB indexes and many other approaches lack parallel computation. The massive amount of spatial data requires a lot of computation and traditional methods are insufficient to address these issues. Other existing parallel processing approaches lack in load-balancing of parallel tasks which leads to resource overloading bottlenecks.

In this thesis, I propose novel spatial partitioning techniques, *Max Containment Clustering* and *Max Containment Clustering with Separation*, to create load-balanced *partitions* of a range query workload. Each partition takes a similar amount of time to process the spatial queries and reduces the response latency by minimizing the disk access cost and optimizing the bitmap operations. The partitions created are processed in parallel using cSHB indexes. The proposed techniques utilize the block-based organization of bitmaps in the cSHB index and improve the performance of the cSHB index for processing a range query workload.

DEDICATION

To my family and friends

ACKNOWLEDGMENTS

"Research is creating new knowledge." - ***Neil Armstrong***

I would like to take this opportunity to express my deep gratitude to my advisor, Dr. Kasim Selçuk Candan. Thanks for providing an excellent opportunity to work on a research and other projects at Emitlab. This was the most challenging work of my academic career. Your guidance and constant feedback has always helped me to achieve the targets.

I also want to thank Dr. Maria Luisa Sapino and Dr. Hasan Davulcu for being part of my thesis committee and providing their valuable feedback.

Most important I want to thank my dad, Vasant Gadkari and my mom, Sangita Gadkari for providing me all the opportunities for education. This would have been impossible without your constant support.

I would like to thank former lab member, Dr. Parth Nagarkar for his valuable inputs on the research. I would like to thank my lab-mates at Emitlab - Yash Garg, Maolin Li, Xinsheng Li, Hans Behrens, Xilun Chen, Shengyu Wang, Sicong Liu, Manjusha Ravindranath, Silvestro Roberto-Poccia, Jung Hyun Kim for their insightful discussions.

I would also like to thank all my friends for helping me in every situation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Background and Motivation

Large number of people use cellphones, gps-enabled devices that tracks the location. The location contains latitude, longitude value. This can be stored easily using point(x,y coordinates) in the database. Most common type of queries on such databases are range queries, join queries, KNN queries. Traditional database management systems use B-Tree, B+ Trees as indexing mechanism for fast lookup. They lack the efficiency for processing spatial range queries. The spatial index structures are used to store the spatial data and later retrieve it effectively. One such example of indexing the spatial objects is Compressed Spatial Hierarchical Bitmap (cSHB) Indexes.

In cSHB Indexes[23], the spatial object is stored using bitmap indexes. The bitmap resolution is determined by splitting the region into equal size quadrants using MX-QuadTree. The bitmaps in each region are compressed and stored in the hierarchical format by Z-order curve in the MX-quad Tree. This creates a hierarchy of 4-ary tree. Refer figure 1.1 for hierarchy creation using MX-QuadTree and Z-order Curve. To gain efficiency for disk access, the cSHB organizes the nodes into blocks of K size. To form a block, the nodes must be at same level and total size of nodes should be less than equal to K size. For querying the spatial range queries are processed using novel cut(node) selection[22] techniques- Inclusive Cut Selection, Exclusive Cut Selection. The blocks required for answering the query is brought into memory (disk access) and bitmap operations (CPU operations) are performed. This reduces the time to query

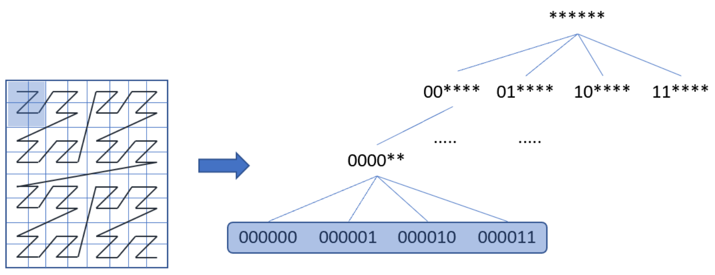

Figure 1.1: cSHB Index Hierarchy Overview. Source : [23]

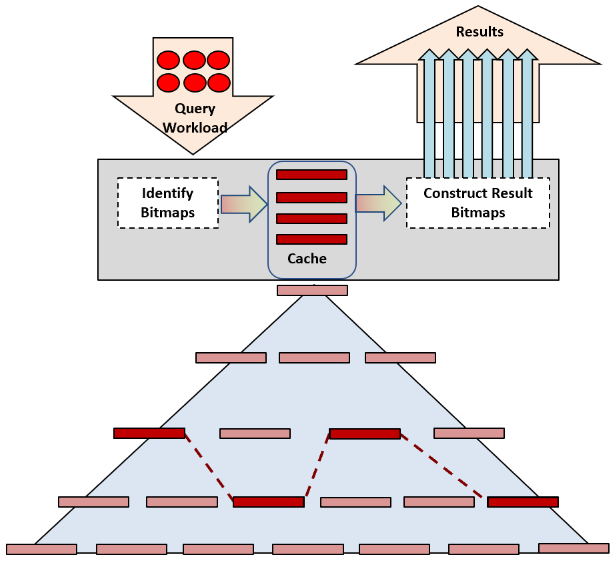

Figure 1.2: cSHB Querying Overview. Source : [23]

the range query workload. For more details on cSHB, refer 3.1 for more details.

The cSHB indexes can perform better if the workload is partitioned in such a way that it leverages the block based hierarchy of the cSHB. This can reduce the disk accesses and CPU operations. More performance boost can be added to cSHB by overcoming the limitations of the query processing discussed in next section.

1.2 Limitations of cSHB Query Processing

The cSHB Index structure performs well when compared with traditional index structures. It also proposes a novel technique using bitmap based indexes to index and process the range query. But, cSHB cannot process large query workloads in parallel. Following limitations are observed in query processing in CSHB:

1. **Large Query Workloads**: cSHB focuses on optimizing and processing a query workload and cannot perform best when large workload are encountered. The computation power of a single machine can limit the processing of large query workloads. To ensure the data gets processed quickly, there is a need of parallel computation.

2. **Parallel computation**: The parallel processing of query workload can increase the performance of the cSHB. Each parallel task can use the cSHB index and provide results much faster than the query workload processing system on single machine.

1.3 Partitioning

Partitioning the query workload allows to process each partition in parallel. The computation power of multiple machines to process a workload can increase the performance of workload processing.

Following are the drawbacks observed in random partitioning:

1. **Redundant Computation**: The overlapping queries have a redundant data and should be clustered together to take the advantage. The performance gain also depends on the degree of overlap to some extent.

   The cSHB uses the block-based organization. Thus, nearby non-overlapping

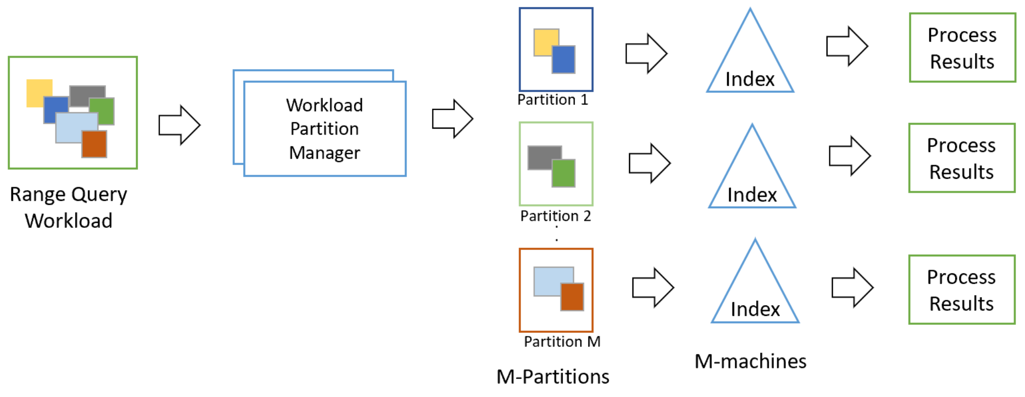

Figure 1.3: Partitioning query workload overview

queries can also have performance boost in the disk access cost of cSHB.

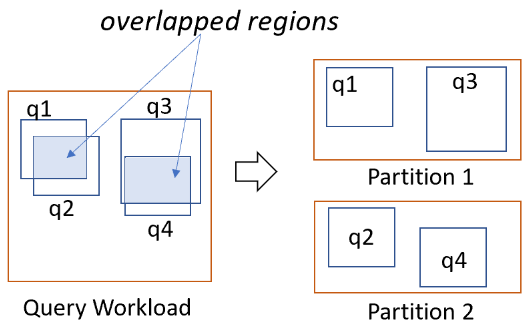

Figure 1.4: Overlapped region example

2. **Unbalanced Partitioning**: The partitions created to compute the queries in parallel can become a bottleneck if one of the partition does not perform well. This can lead to higher response time. Consider an example of 4 partitions, where first 3 partitions perform very fast in 100ms and 4th partition takes 1000ms then the overall performance gets affected by 4th partition.

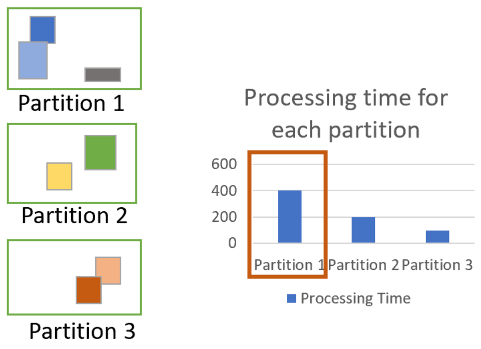

Figure 1.5: Unbalanced partition example

1.4 Problem Statement

Vision: The spatial index structures are used to store the spatial data and later retrieve it effectively. The amount of spatial data generated is very large[29][15][23][6] and requires more time to query. The spatial data query workloads can perform better if they are partitioned such that there will be less access and processing cost required for underlying index structure. Consider cSHB index structure. The query workload contains set of queries that has certain degree of overlap. The query workload can be partitioned by using the overlapping region information to determine the approximate cost of processing the queries. Partitions created should follow these properties-

1. **Load balanced partitions**: Each partition must be load balanced. Ineffective partitioning can lead to bottleneck in performance of the query workload. Parallel processing systems often suffers from unbalanced partitions.

2. **Low Disk Access**: The cost of reading the blocks must be lowered. This can be achieved by increasing the neighboring queries in same partitions. The neighboring queries will select the sibling nodes and the chances of fetching sibling nodes in same block are higher.

3. **Optimized CPU Operations**: To optimize CPU cost in cSHB, the bitmap operations cost needs to be reduced. The number of blocks selected must have a minimal of operations required for fetching the result.

To summarize the problem-

Statement: *Given M machines and range query workload Q, generate M-partitions ($P_1, P_2, ..., P_M - 1$) such that all queries in a query workload belong to only one partition(hard clustering) and all partitions take approximately same amount of time to process the set of queries belonging to the partition.*

The partitions created should follow these properties-

1. Each partition should be load-balanced to avoid resource overloading.
2. The partitions should lower the disk access cost (read bitmap in cSHB)
3. The partitions should optimize the CPU operations (AND, OR, ANDNOT bitmap operations in cSHB)

Method: The partitioning approaches used for creating load-balanced partitions use the spatial data structure(R*-Tree). This allows to group the neighboring queries in a tree structure. The costs are assigned to each level of R* Tree. Based on the approaches, the partitions are created that are processed in parallel using the cSHB indexes. More details for hypothesis and research contribution in next section. Refer proposed approach for details about partitioning approaches.

1.5 Hypothesis

Based on the cSHB index hierarchy and sample query workload statistics, few attributes are considered for boosting the performance. The query workload after creating partitions will improve performance of the cSHB due to following attributes-

1. **Parallelism of task**: The partitions created can run parallely and improve performance of the cSHB query processing.

2. **Load-balancing**: The load-balanced partitions ensures that the time taken for each workload to process is similar. This reduces the chances of resource overloading.

3. **Block-based organization**: The partitions can improve performance by ensuring that neighboring queries are in same partitions. The hierarchy of cSHB indexes considers sibling nodes to be allowed in same block. Thus, it reduces the disk IO as one block will fetch one or more neighboring queries associated with other nodes in that block.

4. **Degree of Overlaps**: Overlapping region improves performance by providing less block accesses. Thus the queries with maximum containment are considered for clustering. It also forms compact clusters that has less empty regions in R* tree making it easier for creating partitions.

5. **Optimized bitmap operations**: The operations on bitmaps are optimized by lowering the block reads. In some cases, after running query statistics, it was found that separation of neighboring queries slightly increases the disk IO (block reads) but improves the CPU cost (bitmap operations) in cSHB. It was observed for queries with overlapping region less than 10%. Thus separating the queries based on separation criteria(here <10% overlaps), the performance of overall partitions is improved.

All these attributes help in contributing to the two approaches to improve efficiency of cSHB.

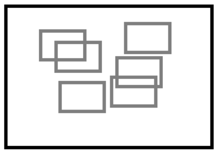

Figure 1.6: Range Query Workload Example. Source : [23]

1.6 Research Contribution

The main contribution of this research is partitioning the range query workload such that it can be processed parallelely and efficiently.

Definition 1.1. ***Spatial Range Query***, *q, is defined by a range that query covers using south-west and north-east pair of points such that $q_{sw}.x <= q_{ne}.x$ and $q_{sw}.y <= q_{ne}.y$.*

A range query consists of range in 1D, 2D or high-dimensional spaces. For example, consider a rectangle. It consists of left, top, right, bottom coordinates. Such rectangle can be considered as a range query in the 2-D spaces.

Definition 1.2. ***Spatial Range Query Workload***, *Q, is defined as a set of range queries with varying range size.*

$$Q = [q_1, q_2, q_3, ..., q_N] \tag{1.1}$$

where, q is the corresponding range query and N is the total number of queries in the set. Figure 1.6 illustrates an example of Range Query Workload.

A Spatial Query workload contains multiple spatial queries. The workload may vary in size and also the range of the queries in Q may be different. Query workload size Q_{size} is not fixed.

Definition 1.3. ***Partition Set*** *P, is defined as the set of partitions of the Query workload Q. The workload Q is divided into M number of partitions $(p_1, p_2 \ldots .p_M)$ such that each of the partitions p_i consists of set of queries from query workload Q.*

$$P = [p_1, p_2, p_3, ..., p_M] \tag{1.2}$$

where, p_i is the partition and M is the total number of partitions in the set P.

$$Q = [p_1 \cup p_2 \cup p_3 \cup .... \cup p_M] \tag{1.3}$$

where, p_i is the partition and M is the total number of partitions in the set P and Q is the query workload

$$\phi = [p_1 \cap p_2 \cap p_3 \cap .... \cap p_M] \tag{1.4}$$

where, p_i is the partition and M is the total number of partitions in the set P

The queries are not repeated in any of partitions. Size of each partition may vary. Refer equations 1.2, 1.3, 1.4 to understand the properties of partitions created from query workload.

A range query workload consists of many queries on the same database. The research intends to process such range query workloads efficiently. Range query workload consists of different range size queries. This research will find the optimal way to partition the set of range queries into *partitions.* These multiple partitions will be processed in parallel. The partitions created are processed on the cSHB index structure to check the performance of the system. The query workload partition manager creates partitions that has less disk access cost and optimized bitmap operation cost. In parallel computations, the processing speed varies according to the partition. The results for computation may be inefficient if any of the partition does not perform well. If the partition is not load balanced then it creates a bottleneck and affects the overall processing time. In order to avoid such unbalanced clustering of queries, the new techniques to partition workload considers the approximate cost to process the workload. These techniques ensures the load balancing such that each query partition processing time is similar. Each partition in the multiple query partitions has a similar cost. The two approaches/techniques discussed in proposed approach are:

1. Max Containment Clustering (MaxCC): Creates partitions by clustering the nearby queries.

2. Max Containment Clustering with Separation (MaxCC-S): Creates partitions by assigning queries to a separate cluster based on the separation criteria.

The first approach creates load-balanced partitions that reduces the disk IO by lowering the block reads. The second approach creates load-balanced partitions that optimizes the CPU operations by minimizing the bitmap operations. The proposed approach is discussed in chapter 4.

1.7 Organization of Thesis

The content of the thesis is organized as follows:

- Chapter 2 focuses on the Background and Related work

- Chapter 3 describes the Spatial Index structures (cSHB, R-Tree and other variants, clustering)

- Chapter 4 introduces and explains the Proposed Approach. This chapter contains details for the query workload partitioning algorithms.

- Chapter 5 contains the Experiment details. It contains information for experimental setup, data set, evaluation criteria and performance of the system.

- Chapter 6 concludes the thesis and describes the direction for future work.

Chapter 2

RELATED WORK

2.1 Existing Approaches and Challenges

Single query processing system: The spatial database management system mainly focus on optimizing the single query execution. Therefore they cannot leverage other query information to get the results. There has been a lot of research to efficiently process spatial queries such as kNN, spatial join queries. Due to compute intensive nature of the spatial data, the queries require a lot of overhead. The single query processing system cannot take advantage of parallel or distributed execution of queries. Parallel (Spatial database management system)SDBMS systems[5] are developed that partition data to multiple parallel disk to reduce the I/O bottleneck. But these systems are not fully capable of handling large number of spatial queries efficiently.

Indexing and Querying challenges in distributed environment: Many other approaches to process spatial data use big data management and processing systems such as Hadoop[11][12], Spark[30][30]. Few systems implemented use geographic proximity[10] of spatial objects along with a two-tier distributed spatial index to prune search space. Hadoop based frameworks such as SpatialHadoop[12] and Hadoop-GIS[5][27] are used to process queries in parallel. Geospark[4], SpatialSpark[28] is built on top of Spark to support geospatial operations. All these systems support different spatial queries such as KNN, range, join queries. To maintain the indexing structure is distributed environment is very difficult. Most of the systems[12] use two layer indexing in distributed environment. The global layer indexes the coarsegrained information and local layer on each node organizes the actual data. This makes sure that indexing gets aligned with Map-Reduce programming paradigm[11]. The global index acts as driver(main) program and initiates mapreduce task and local indexes processes map tasks. In the paper[31], the new spatial database management system called VegaGiStore is introduced. It also uses two-tier bases spatial indexing approach. Similar to earlier global and local concept, it tries to improve the query computation using MapReduce.

Unbalanced loading of parallel tasks: The common drawback of the approaches (mentioned above) is the underutilized resources due to unbalanced parallel tasks. The computation cost for a system with unbalanced workloads is expensive and affects the overall performance of the system. Their inability to scale and process data in parallel becomes a problem. The unbalanced tasks can create resource overloading bottlenecks. Load balancing is a challenging issue and needs to be carefully addressed when designing such parallel processing system. To predict the time taken for query to execute is very hard. The approximate prediction can be helpful. This may based on the insights from previous queries executed on the same underlying data. In the paper "Efficient Spark based Framework"[6], the service layer is introduced on top of Geospark. This service layer forms unbalanced clusters using a DBSCAN-MR algorithm and then self-adaptation service at each attempt calculates new configuration of cuts to balance timing in partitions. This approach takes many iterations to find the cutting factor that balances the local execution time in all partitions [6].

To tackle problems/challenges mentioned above, the new system is proposed that processes the query workload in parallel and introduces load balancing to improve performance.

2.2 Need of Spatial Partitioning

Traditional DBMS lack in processing the queries efficiently for a large set of geospatial data. Some approaches apply distributed indexing concept that distributes data along the distributed structure at indexing[5]. Spatial databases mostly use R-tree, quadtrees and other variants of spatial data structures[7] for fast access. Parallelizing R-tree indexes is a challenging task as the assumption that tree is always balanced is incorrect. Quadtrees[24] may seem an efficient solution for creating balanced trees but are ineffective in case of skewed data.

To overcome the challenges mentioned earlier, there is a need of spatial partitioning that helps in processing data efficiently using underlying index structure. The Compressed Spatial Hierarchical Bitmap (cSHB) Indexes use Z-curve to impose a hierarchy on space and preserve data locality[23]. The cSHB Indexes were built to process multiple range queries on a single machine[23]. To improve performance cSHB uses block-based organization.i.e. sibling nodes at the same level are stored together in compressed form. The hierarchy of bitmap indexes allows different ways or query plans to answer the given query[22]. The cSHB takes more time to process large query workloads as it executes on a single machine. It cannot take advantage of the block-based organization to execute a large number of overlapping multiple range queries. A parallel processing system can handle this issue and compute results even faster than existing cSHB. To process data in parallel, the system needs to partition tasks effectively. Clustering algorithms can be used to group the similar type of queries. This can improve performance in querying millions of spatial objects.

Chapter 3

SPATIAL INDEX STRUCTURES

3.1 cSHB - Compressed Spatial Hierarchical Bitmap Index Structure

Compressed Spatial Hierarchical Bitmap(cSHB) index structure was introduced in 2015 [23]. Compressed Spatial Hierarchical Bitmap(cSHB) index structure is used for efficient processing of the range query workloads[23]. It creates bitmaps for point objects based on the regions that are determined by MX-quadtree. To create a spatial hierarchy, Z-order curve is used at each level in the MX QuadTree.

- **Bitmap Index**: Bitmap Indexes contain bit arrays with 0's or 1's as their values. Bitwise logical operations on bitmaps are required for finding the points belonging to that specific region. Most common operations on bitmaps are AND, OR, XOR. During the query access, the bitmap-indexed columns are combined using the operators to provide the results. Refer figure 3.1 for bitmaps for points in space divided into two regions.

- **QuadTree**: A tree-structure that divides region into 4 quadrants. Each cell/quadrant has some data points. If the size of cell is greater than maximum capacity then the node is further split into four quadrants. It creates 4-ary tree structure.

- **MX-Quad Tree**: A variant of Quad tree which splits the cell always at the middle. Example- refer 3.2

- **Z-order curve**: The Z-order curve is used to convert multidimensional data to 1-D data. It imposes the hierarchy in space. Example for Z order curve in $2^h * 2^h$ space MX-Quad Tree, h=3 (refer 3.2).

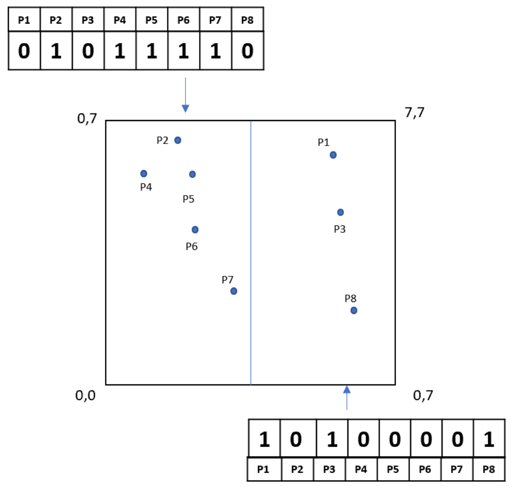

Figure 3.1: Bitmaps for the given points in space divided into two regions. Source:[23]

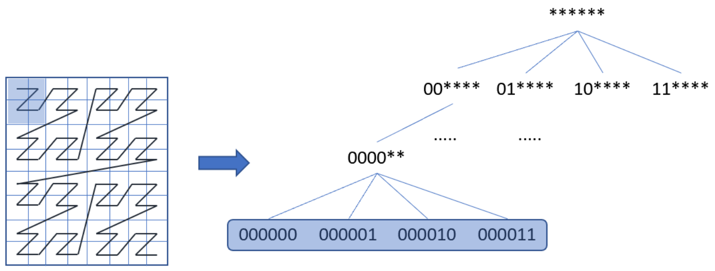

Figure 3.2: Z-order curve hierarchy. Source:[23]

Based on the hierarchy, nodes are combined together to form the parent node. This merging creates a 4-ary tree structure and forms a spatial hierarchy. (refer 3.2).

- **Spatial Hierarchy**: The cSHB [23] associates space S a hierarchy H that contains node set $N(H)$ = set of all nodes
  - Node: A node belongs to set of all nodes. Each node has a bounded region in space.

  - Parent and children node: A parent node N_p is considered as parent of nodes(n_i) if all the nodes(n_i) has only one and common parent N_p. Such nodes(n_i) are all the children nodes.
  - Internal nodes: Nodes in the hierarchy that are not at the last level of the hierarchy are Internal nodes.
  - Leaves: Nodes that belong to last level in hierarchy are the leaf nodes in the hierarchy.
  - Sibling nodes: The set of nodes n_i that has a common parent then all the nodes in n_i are siblings of each other. Hierarchy of bitmaps helps to answer a query of different sizes.

  Bitmaps are compressed and stored for each node (leaves, internal nodes). Compression helps in reducing the size of the node. To reduce the IO cost of compressed bitmaps, the cSHB introduces block-based organization of bitmaps generated. The sibling nodes at same level are added to same block. This reduces the IO during query processing as the neighboring bitmaps might be useful for query plan.

- **Blocks**: Assume K as the target size of blocks and B_i as compressed bitmaps of nodes at same level. Block is formed by concatenating bitmaps ($B_1 \ldots B_n$) at same level until they are less than target size K. Example- Consider following Bitmaps at same level $< Bi, size >$:$< B1, 5 >, < B2, 4 >, < B3, 2 >, < B4, 2 >$

- **Cut**: A cut C is a subset of internal nodes of the hierarchy H, relative to a workload Q, satisfies validity and completeness conditions[23][22]. Validity: For relevant hierarchy, there is exactly one node in cut for any root-to-leaf

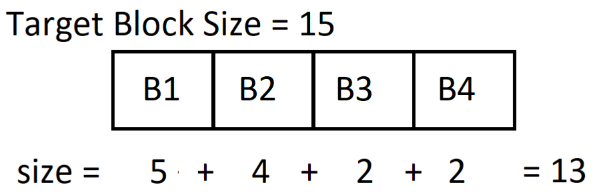

Figure 3.3: Block based organization in cSHB. Source:[23]

branch. Completeness: The nodes in cut covers all the leaf nodes in the relevant hierarchy.

- **Query Plan**: The spatial hierarchy created using above method is used for generating efficient query plans for range query workload. There are following steps involved in generating effective query plan and fetching the nodes(cut) required for answering the queries efficiently.

  Node cost estimation: In this step, relevant hierarchy(R) nodes are assigned the cost. For this, all the nodes belonging to at least one query in query workload (Q) are identified in top down manner[22]. This relevant hierarchy is assigned the cost in bottom-up approach.

  Steps:

  - **Top-Down Traversal and Pruning**:

    Starts from the root and finds the nodes (n_i) that satisfies any query (q) from the query workload(Q).

  - **Cost Computation**: The relevant hierarchy(R) contains internal nodes that are visited in bottom-up order to assign the cost estimate for processing each query. **Cost estimation and Leaf Access Plan**: Using the following two strategies the cost estimate is computed. Then based on the best cost estimate the query plan (inclusive or exclusive) is decided for the query. **Inclusive Query Plan:** If query uses this strategy to provide results then all the leaf bitmaps will be identified belonging to the query and using bitwise operations(OR) the result will be provided.

The cost estimate for this is the sum of size of bitmaps identified for combining and providing results. **Exclusive Query Plan** If query uses this strategy to provide the results then a bitmap(B_i) that belong to the query is identified and the bitmaps(B_j) that if excluded from that query are identified and using bitwise(ANDNOT) operation on set of such bitmaps will provide results for the query.

The cost estimate for this is the size of Bi and sum of size of B_j.

- **Cut Bitmap Selection**: After the cost assignment step, the cut (nodes) selection process traverses the hierarchy in bottom-up order.

  In order to select the optimal cut, the process depends on the estimated cost and block IO (described earlier). If a block(b1) is accessed for any query in the workload earlier then the cost of blockIO is not included.

  Cut Bitmap selection process checks the cost of node (hybrid cut cost + blockIO) is less than or equal to the cost of children(sum of chidren nodes hybrid cost + blockIO). If the condition is satisfied then the parent node is selected to form a cut for providing results. formula

3.2 R-Tree Index Structure and its variant

R-Tree is a spatial data structures widely used for indexing multidimensional data[13]. R-tree node contains MBR(minimum bounding rectangle) information[2]. MBR represents the covered area by the spatial objects inside the node. R-Tree has properties similar to B-Tree. The R-Tree is also balanced search tree. The MBR helps to make decision starting from root to traverse down towards the object. Finding object becomes easier as the MBR allows to determine which subtrees need to be traversed. Most common tree searching algorithms[13][10][20] such as containment, intersection, nearest neighbor are very simple to perform using R-Tree structure. Example of R-Tree- Refer figure 1

R-Tree uses linear and quadratic node splitting algorithms. If the node overflow occurs while insertion, these algorithms are used.

Drawbacks of R-Tree: Large number of empty regions creates large MBRs. This leads to overlaps in MBRs and more nodes/sub trees need to be accessed. The query performance drops down a bit based on the empty regions and overlaps in MBRs. R*-Tree (discussed in next section) solves this problem.

R*-Tree: R* Tree is a variant of R Tree. It provides less node overlaps and less storage requirements[25]. The major change is R* Tree is the node split policy. This policy tries to minimize the coverage and overlaps[9][25][18]. The results using R* Tree has very compact boundaries and produce better results and less overlaps. Example-Refer fig

Three major changes in R* Tree algorithm[25]

- Intelligent object insertion procedure
- Re-insertion of objects in overflow node
- Overflow node split manner

3.3 Other Spatial Data Structures

B-Trees, BST, AVL Tree are all effective in handling one dimensional data. They are not sufficient to process the spatial queries such as range queries[16]. Spatial data structures are used to handles such data efficiently.

- kd Tree: Extension of BST to multiple dimension. The splitting decisions in kd tree are alternate among every dimension.
- Quad Trees: A point region quad tree that splits the region into 4 quadrants.It has a 4-ary tree structure.

3.4 Clustering

The notion of "cluster" is not precise, thus different algorithms/models exist for different scenarios. Different models exists in order to understand the differences in the algorithms[26][8][21][17].

- Hierarchical Clustering: Clusters are formed based on the distances between the objects. Algorithm Examples- Single Linkage, Complete Linkage Clustering
- Centroid based Clustering: Based on centroid and object distances.[14][1] Algorithm Examples- K-means Clustering
- Distribution based clustering: Objects belonging to same distributions form cluster. Algorithm Example- Gaussian Mixture model (using expectation maximization)
- Density based Clustering: Algorithm Example- DB-Scan Algorithm

Chapter 4

PROPOSED APPROACH - QUERY WORKLOAD PARTITIONING

4.1 Research Challenges

The query workload partitioning is able to create partitions that are executed on parallel machines. But, there are few drawbacks related to random partitioning-unbalanced partitions and redundant computation (discussed in section 1.3). These provides the following research challenges-

1. Does individual query workload partitions take similar processing time?

2. How to avoid redundant computations of overlapped regions?

I have focused on providing solution for these research challenges.

4.2 Contribution

4.2.1 Research Questions

Based on the research challenges discussed in previous section 4.1, following questions are important to solve in order to create partitions.

1. Can workload partitioning leverage proximity between queries?

2. Can bitmap access cost be reduced using block-based cSHB hierarchy?

3. Is it possible to optimize bitmap operation cost in cSHB?

The queries that are close to each other needs to be grouped together. Consider the grouping scenario shown in the figure 4.1. All the scenarios have different performance. The query sizes are same in each scenario. The only difference is the

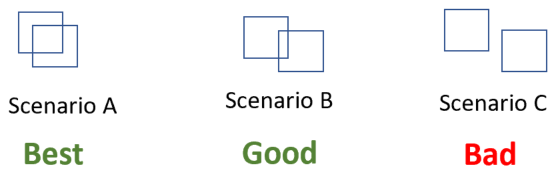

Figure 4.1: Performance for different groupings

proximity of the queries. In scenario A, the overlapped region is more so less number of computations are involved. In scenario 2, very less overlap so it performs good but not the best as compared to scenario A. In scenario C, there is no overlap. Thus the performance of scenario C is very bad. Based on different grouping scenarios, the groupings with maximum overlap perform better than other groupings.

4.2.2 Clustering nearby queries

The primary intuition is to group nearby queries in same partition. To solve this problem, a tree structure can be used. At leaf level the queries can added and intermediate node marks the grouping of the queries. To group the queries for best performance, following additional constraints should be imposed-

1. More overlaps in the queries inside a node
2. Less or no overlaps across the nodes

Refer figure 4.2, that shows 2 different grouping for query workload. The first grouping creates the nodes containing overlapped queries (first constraint satisfied). It also maintains the less overlap constraint between the nodes in a tree. The second grouping fails to satisfy both constraints. To create partitions using a tree, branches are assigned to different partitions. In first grouping, the partitions created has overlapped queries in same partitions. In second grouping, the partitions created has

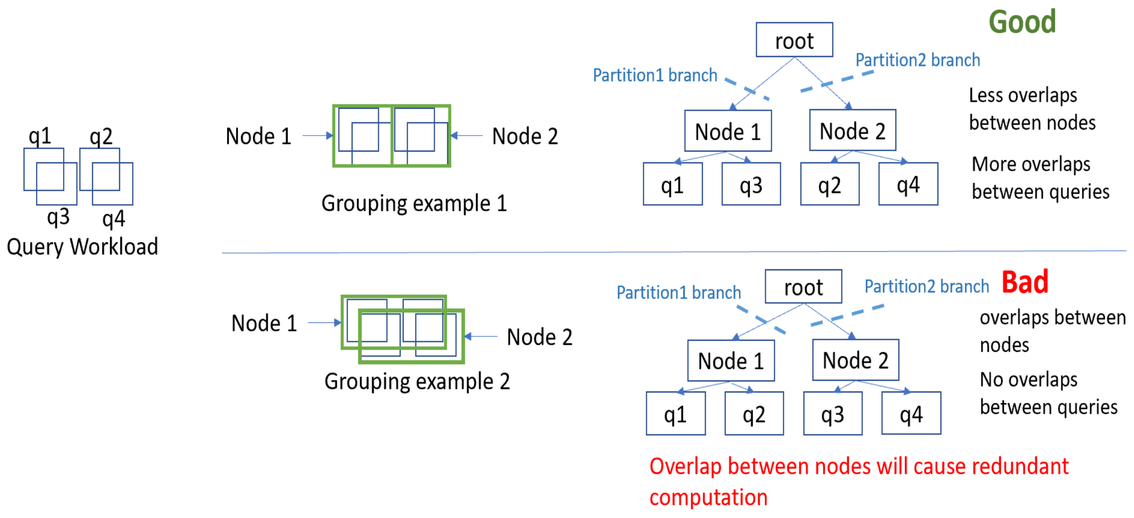

Figure 4.2: Effect of groupings on the partitions

no overlap. This is due to the failure to satisfy the constraints. The queries with overlapped region get separated and results in redundant computation for overlapped regions. The solution to create the groupings with minimum bounding region for the nodes is to use spatial R* Tree for grouping the queries.

4.2.3 R Tree*

The R* Tree provides good results for clustering the 2D objects. The other spatial data structures are not able to create tree hierarchy using range queries that has compact MBRs than R* Tree. The R tree also creates hierarchy but not as good as R* Tree. The R tree MBR contains empty regions and also overlaps between other MBRs. At the time of insertion of queries, the node splitting algorithm in R* Tree[3] re-inserts all the objects inside the overflow node. This makes sure that all MBRs formed have less empty regions and less overlaps as well. The details about Spatial data structures and drawbacks is discussed in Chapter 3.2.

I checked the results for R-Tree and R*-Tree on sample query workloads. The

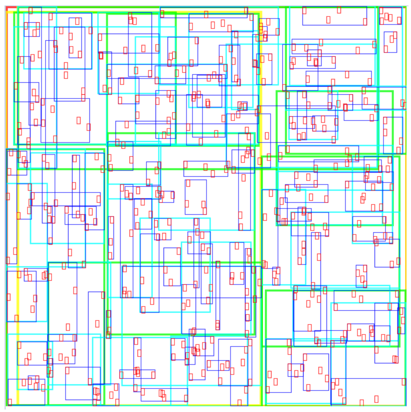

Figure 4.3: R Tree with max children=5 and 500 range queries (in red color) and MBRs created at different levels.

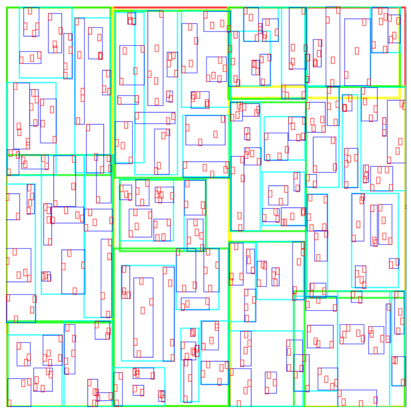

Figure 4.4: R* Tree with max children=5 and 500 range queries (in red color) and MBRs created at different levels

R*-Tree[4] forms more compact MBRs than R-Tree. Example- For 500 queries and max children=5, the R-Tree vs R*-Tree comparison can be found in figure 4.3 and 4.4 respectively.

The R*-Tree[4] constructed for a query workload consist of hierarchy R that contains actual queries from workload at the last level. All the other levels from root to second last level contains MBR information. The MBR information of a node N specifies the region bounds using $N_{sw}.x$,$N_{sw}.y$ and $N_{se}.x$,$N_{se}.y$. Refer figure 4.5 for

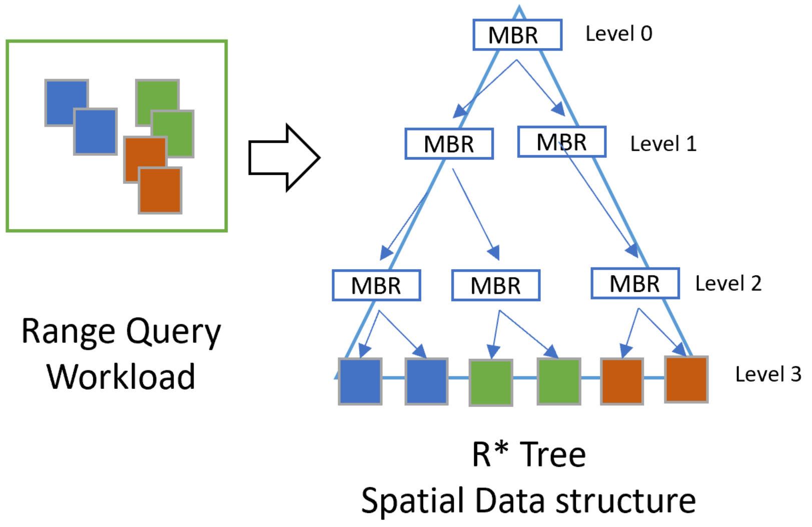

Figure 4.5: R*-Tree Example for 6 queries and Max children=2. Tree shows the MBR nodes and leaf nodes containing queries

R* Tree hierarchy.

Additional information is also computed for each MBR node. The list of parameters calculated for each node are explained in next section 4.2.4.

4.2.4 Cost Computation

- **Initial Cost** of a node C: Cost of processing all the queries in the node.

$$C = \sum_{i=1}^{N} c_i \tag{4.1}$$

  where, N is the number of queries in the node and c cost of processing each query. The cost c_i depends on size of query q_i.

- **Projected Cost** C_p: The approximate cost to process the query with respect to neighboring queries overlap and distance. This takes into account the discount and tax for the initial cost.

$$C_p = C - D + E \tag{4.2}$$

where, D is the discount based on overlap degree and E is the empty are tax.

Example- Projected cost of leaf level query will be same as intial cost. But when it is grouped with other neighboring queries then it will get discount and tax cost added. The discount and tax is explained below.

- **Discount** D: The region that intersects with other queries is called the overlap region. Based on the overlapped region, a discount should be provided for the node. Example- Refer figure 4.6 that shows overlapping region for 2 queries. The overlapped queries has performance gain so it should provide discount on initial cost of processing the queries.

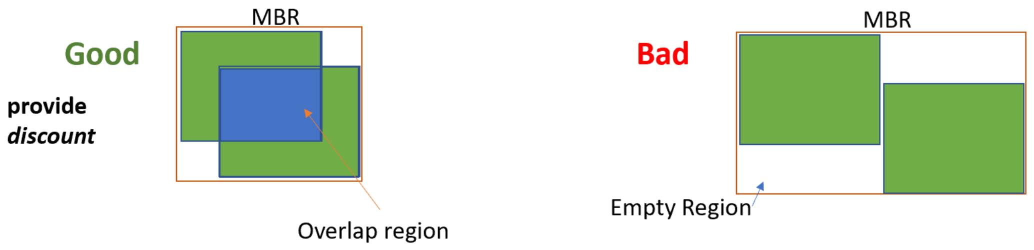

Figure 4.6: Discount based on overlapping region in queries.

- **Tax** E: Tax is based on Empty Area of the MBR node that is not overlapped with any query. More the empty area, more the cost of processing query. Refer figure 4.7.

The cost is computed only at the second-last level in the tree. For rest of the intermediate nodes the cost propagation takes place in bottom-up manner. This reduces the

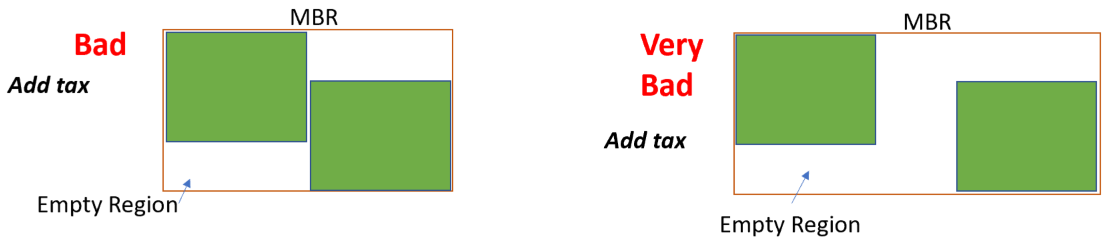

Figure 4.7: Tax based on empty region in queries.

complexity to predict the cost for all the nodes. More details on cost computation with example are explained in the next section 4.4.

4.2.5 Create Partitions from a Tree

The cost computation assigns cost to every node in a tree. Based on this cost, the branches can be traversed for queries to assign to a partition. Each partition has the expected budget. In order to fill each of the partition with queries, a tree traversal takes place. On the way it adds the queries to first partition. Once the expected budget of a partition is reached, the next queries get assigned to second partition. This process continues till all queries get assigned to at least one partition. Refer figure 4.8.

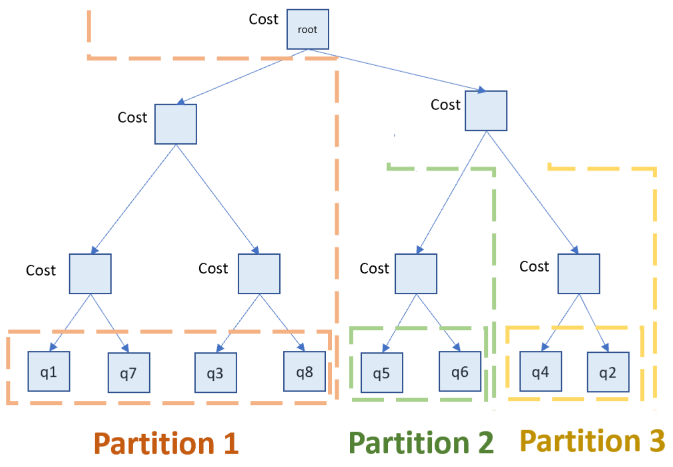

Figure 4.8: M-Partitions from a tree structure

4.3 Architecture Overview

The range query workload consists of varying size queries. This workload contains overlaps between queries. To parallelize the processing of query workloads, multiple partitions should be generated. Each query should belong to only one of the partitions. Refer 1.2 for equations and more information on partitions. Creating partitions is similar to creating clustering. The additional constraint imposed on “partitions/clusters” is that all of the clusters should be load-balanced. Thus to group the range query workload there is a need of using the spatial data structures that forms grouping/clustering as per the requirement.

Figure 4.9 provides the overview of the architecture to partition the workload and further process these partitions using cSHB indexes. The process takes the query workload as an input and based on the workload generated spatial R*-Tree structure. It then uses one of the approaches for created workload partitions using the partition manager on top of spatial data structure generated earlier. The partitions in the partition set are then processed parallely using cSHB index structure to provide

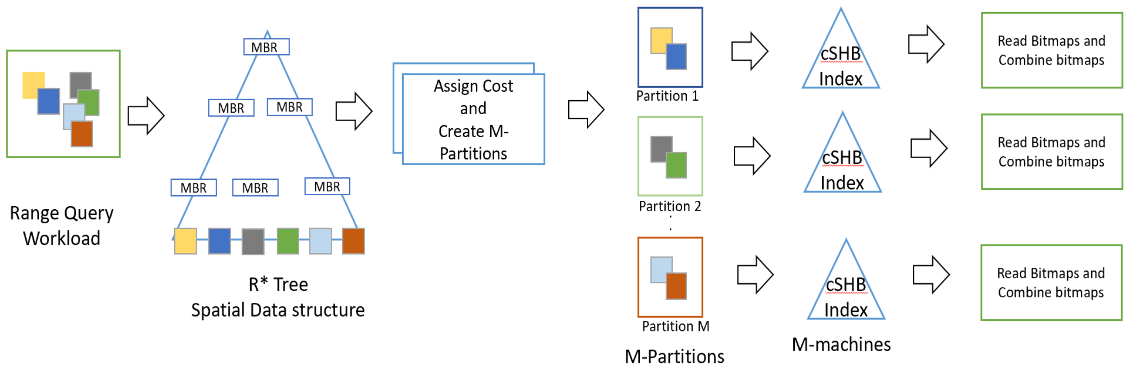
Range Query Workload
MBR
R* Tree Spatial Data structure
Assign Cost and Create M-Partitions
Partition 1
Partition 2
Partition M
M-Partitions
cSHB Index
M-machines
Read Bitmaps and Combine bitmaps

Figure 4.9: Architecture overview for partitioning range query workload and processing partitions using cSHB indexes.

results using bitmap operations. The query statistics helps in making decision to what extent the overlap is beneficial for the workload.

The following sections discuss the two approaches to partition range query workloads.

4.4 Approach 1: Max Containment Clustering (MaxCC)

4.4.1 Motivation

The *block-based structure* in the cSHB index can take advantage of the neighboring queries. For example, if two query ranges belong to sibling nodes in the cut, then it helps in minimizing the number of blocks fetched. This results in low Block IO (disk IO) cost. Finding neighboring queries is important to reduce the Disk IO cost for fetching the bitmap blocks.

The range query workload contains overlaps between queries. For overlapping regions, most of the cut(nodes) selected is similar. Using this overlapping region it is possible to create load balanced partitions. Assume, if two queries share some

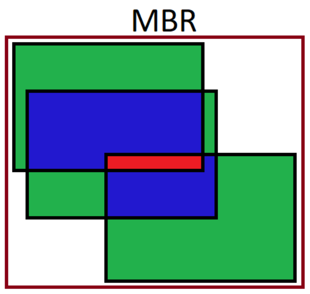

Figure 4.10: Overlapping area for 3 queries.

amount of region, that overlap region will reduce the cost of processing the query. Thus, the cost model takes into account such overlapping regions in queries and provides discount (reduced cost) for processing the queries.

4.4.2 Terminology

- **Overlap Area**: The region that intersects with other queries is called the overlap region. If multiple queries has same overlapping region then it should account for such overlaps. Example- Refer figure 4.10 that shows overlapping region for 3 queries

- **Covered Area**: Area covered by all the queries together. This not the sum of individual queries but the union of area of all the queries. It considers the overlapping region only once. Example- Refer figure 4.10 for overlapping region of 3 rectangles. The covered area is union of all 3 rectangles i.e. area marked by green, blue and red.

- **Empty Area**: Area of the MBR node that is not overlapped with any query. For example. Refer figure 4.10. Consider the white area inside the outer rectangle as Empty Area. It can be computed using following:

$$EmptyArea = MBRTotalArea - CoveredArea \tag{4.3}$$

- **MBR Total Area**: the area bounded by that node. It is the normal area calculation of bounded region of MBR.

$$MBRTotalArea = MBR_{length} * MBR_{height} \tag{4.4}$$

- **Overlap Area Percent** O_p: Based on the overlapping region and total covered region, calculate the percentage of overlap with the query.

- **Empty Area Percent** E_p: Based on the empty area and MBR total area it is possible to get the Empty Area Percent

- **Overlap Discount Map**: This map contains the values for discount based on the percentage of overlap. The discount map is computed using the stats generation module that creates temporary workload with different degree of overlaps and calculates the performance of queries based on overlap degree. This map helps to assign discount to queries based on the overlapping region percentage.

- **Empty Area Tax Map**: This map contains the values for tax based on the empty region. The values for this map are generated based on the stats generation module that runs temporary workload to check till what extent the empty area affects the workload. This map helps to assign tax to queries based on the empty area percentage.

- **Query List**: Internal nodes contain information of their respective leaf level queries. This is used to avoid traversing to leaf level if internal node satisfies the condition of adding to partition. Example- Assume hierarchy H with depth 8, then if node $N3_1$ at level 3 can be added to partition (algorithm described in step B 4.4.3) then to access its leaf queries the traversal cost will be high. To speed up the retrieval of queries, each internal node maintains its leaf level query list.

4.4.3 Steps and Algorithm

Strategy 1 is divided into two steps a) assign costs and b) create M partitions. In first step, to assign costs strategy follows bottom up approach. In second step, the strategy follows top-down approach.

Step A] Assign Costs: *Given R*-Tree Hierarchy H, assign projected cost for each node n_i based on the region covered and overlaps.* In this step, the R*-Tree is traversed using bottom up approach to assign cost. Each internal node is visited once to calculate the projected cost of each node. At second last level the overlapping region and empty area is computed to provide the approximate discount and tax on the initial cost. Based on the overlapping region among the set of queries at second last level, the discount for MBR node is calculated. Similarly, the empty area calculation is performed and the values are fetched to provide the projected cost of MBR node. This projected cost is used for calculating the cost of inner nodes (using bottom-up approach).

Algorithm 1 describes the assign cost process

Example- Refer figure 4.11. The MBR nodes are assigned projected cost from second last level to root level. This cost is used for pruning the branch for creating the partitions in the step B.

Step B] Create M-partitions: *Given Hierarchy H, find multiple partitions $P(p_1, p_2...p_M)$ such that sum of projected cost of all queries in the partition are simi-*

Algorithm 1 Assign Costs bottom-up

```
1: Input: R*-Tree Hierarchy H, OverlapDiscountMap $O_{map}$, EmptyAreaTaxMap $E_{map}$
2: Output: projected cost estimate and additional MBR information for each internal node, $n_i \in H$;
3: Initialize root
4: for all nodes $n_i \in H$ in bottom-up fashion do
5:     if $n_i ==$ "leaf" then
6:         Calculate covered area using query region information
7:     else if $n_i ==$ second last level node then
8:         Access all children and find the overlapping region and empty area in MBR.
9:         $Discount$ = compute-overlap-and-discount($n_i$, $O_{map}$)
10:        $EmptyAreaTax$ = compute-empty-area-and-tax($n_i$, $E_{map}$)
11:        Based on this information calculate the projected cost
12:    end if
13: end for
```

lar.

To create multiple partitions, the given R* tree node hierarchy is traversed in top-down manner. The decision to add the node to current partition is taken by Projected Cost. Budget of each partition must be similar. Therefore to create M-partitions the projected cost of root is taken and divided into equal M parts.

$$B_{exp} = C_P.root/M \tag{4.5}$$

where,B_{exp} is the budget, C_P.root is the projected cost at root, M is the number of partitions

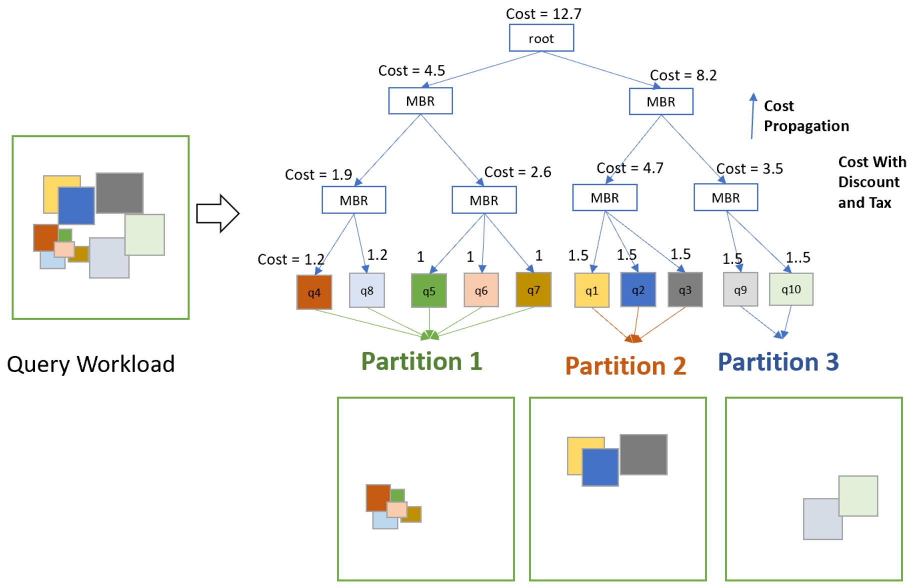

Figure 4.11: Assign cost example using MaxCC

Whenever a query satisfies the current partition budget requirement, it is added to the partition and projected cost for that query is added to the current budget.

Algorithm 2 describes the create partitions process

Example 1- Refer figure 4.11 MBR nodes figure. In this hierarchy, if the M is given 3 then the partitions will have expected budget = (12.7/3) = 4.23 as per equation 4.5. The Algorithm traverses top-down the hierarchy and left of root gets assigned to partition p_1. The right of root is traversed. The cost is more than expected budget of second partition, so it goes down further. MBR node containing $q1, q2, q3$ gets assigned to p_2 and MBR node containing $q9, q10$ gets assigned to p_3.

Example 2: For query workload of 1200 queries of query range 1.5 percent, following are the four partitions created. Refer figure 4.12 Each partition is marked in different colors.

Algorithm 2 Create M-partitions top-down approach

```
1: Input: R* Tree Hierarchy $H$, Number of partitions $M$
2: Output: M-partitions ($P_1 ... P_M$)
3: Initialize root
4: Calculate $B_{exp} = C_p.root$ / $M$
5: Initialize $M$ partitions budget to 0. $B_i \dots B_M = 0$
6: $current_d = 1$
7: for all internal nodes $n_i \in H$ in top down fashion do
8:   if $n_i$.projectedcost <($B_{exp}$ - $B[current]$) then
9:     Add the queries belonging to node in the current partition
10:    $q_i \in n_i, add q_i to D[current_d]$
11:    $B[current_d] += n_i$.projectedcost
12:  else
13:    access all children of $n_i$ one by one
14:    Repeat step 7 for each children
15:  end if
16:  if $B[current_b] >= B_{exp}$ then
17:    $current_b = current_b + 1$
18:  end if
19: end for
```

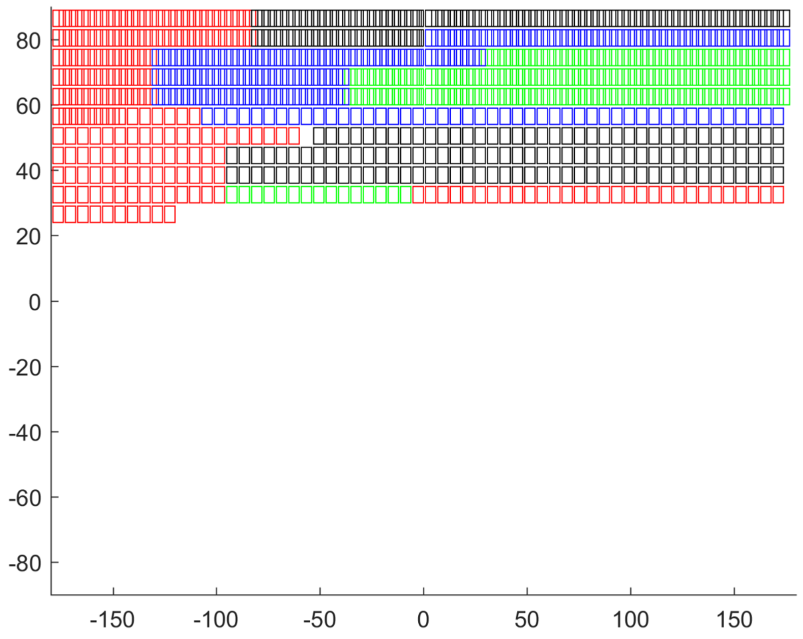

Figure 4.12: Results for query workload of 1200 queries using MaxCC with M = 4.

4.4.4 Conclusion

This two step algorithm generates balanced query partitions. The bitmap IO cost gets reduced as nearby queries are clustered together. Less number of blocks accessed. Almost similar time to process the partitions parallelly. Combine bitmap operation varies for few cases. Overall MaxCC provides good results as per expectations (see Experiments 5.4 in next chapter).

4.5 Approach 2: Max Containment Clustering with Separation (MaxCC-S)

4.5.1 Motivation

The MaxCC may not perform best in few cases. The initial tests on data provided the insights that if there are queries that has less than certain degree of overlaps than combine bitmap operations perform better. To reduce the time taken for CPU

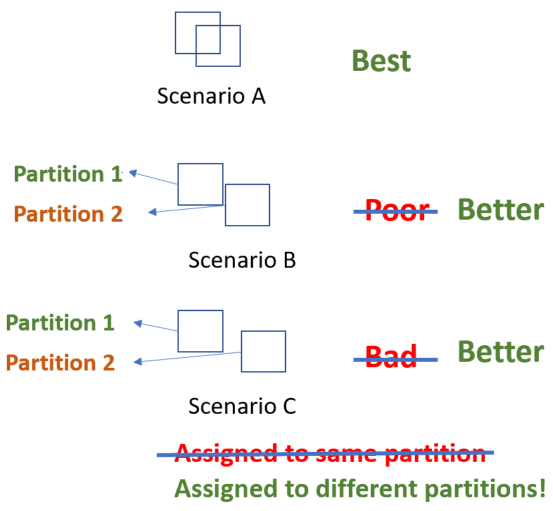

Figure 4.13: Performance for groupings with no overlaps

operations a separation strategy must be introduced. Consider an example of uniform dataset, where less than 10 percent overlap increases the cost of bitmap operations. This as a basis of the max strategy can be used with the R* tree construction to create partitions such that it creates cluster of queries that overlap the most and separate the queries that may have chances of increasing cost. Following approach is used for implementing the MaxCC-S.

This approach has similar flow as MaxCC. First, a R*-Tree is created with additional node information. Then, Assign Cost step assigns each MBR the projected cost and other parameters. One major difference in this strategy is the usage of Split flag as an additional parameter. This Split parameter helps in making the decision to split node into different partitions or to assign node itself to one partition. The definition of split flag is explained in the 4.5.2 section. The usage of Split flag can be found in 4.5.3 Step B of this approach.

4.5.2 Terminology

Separation criteria: This criteria depends on query's performance based on degree of overlaps. In some cases, very few percent overlap is not good. So there is a need to separate such queries. The overlap percent below some threshold becomes the separation criteria.

Split Flag: A boolean value, true or false, determines if the MBR node should be used directly and added to one partition or splitted into different partitions. The decision to split or not is based on the separation criteria. To implement a separation strategy, there is a need to update node MBR with additional information of Split flag(true/false). Consider the example of uniform data set where overlap less than 10 percent is criteria for splitting the node. In such cases the queries that belong to overlap <10 percent will be split and distributed across all the partitions.

4.5.3 Steps and Algorithm

Strategy 2 is divided into two steps a) assign costs and b) create M partitions. In first step, to assign costs strategy follows bottom up approach. In second step, the strategy follows top-down approach.

Step A] Assign Costs: *Given R* Tree Hierarchy H, assign projected cost and split flag for each node ni based on the region covered and overlaps.* This process is similar to the bottom up approach (refer 4.4.3) explained in MaxCC except one change. If the separation decision criteria is satisfied then mark the node split flag as true, otherwise it is false. At higher level, the parent node will access children node split flag values. All values of children are ORed to assign value to parent node.

Step B] Create M-partitions *Given Hierarchy H, find multiple partitions* $P(p_1, p_2, ..., p_M)$ *such that sum of projected cost of all queries in the partition are similar.* To create

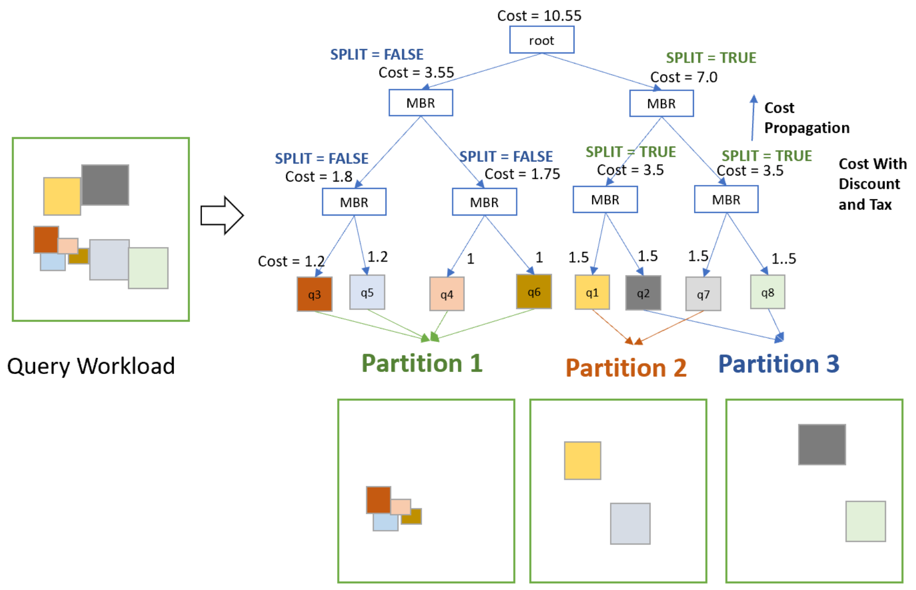

Figure 4.14: Assign cost example using MaxCC-S

multiple partitions, the given R* tree node hierarchy is traversed in top-down manner. The decision to add the node to current partition is taken by Projected Cost and the split flag. If the split node is true then each query is assigned to separate partitions in round-robin fashion. Expected Budget, B_{exp}, is similar to earlier equation 4.5.

Algorithm 3 describes the create partitions process using separation criteria

Example 1- Refer figure 4.14 MBR nodes figure. In this hierarchy, if the M is given 3 then the partitions will have expected budget = (10.55/3) = 3.52 as per equation 4.5. The Algorithm traverses top-down the hierarchy and left node split flag is false so it gets assigned to partition p_1. This assignment completes the expected budget in partition 1. The right child of root MBR has split flag True. Thus the nodes are traversed down the hierarchy to split the queries in rest of the partitions i.e p_2, p_3.

Example 2: For query workload of 1200 queries of query range 1.5 percent, fol-

Algorithm 3 Create M-partitions top-down using separation criteria

```
1: Input: R* Tree Hierarchy $H$, Number of partitions $M$
2: Output: M-partitions $(P_1...P_M)$
3: Initialize root. Calculate $B_{exp} = C_p.root$ / $M$
4: Initialize $M$ partitions budget to 0. $B_i \ldots B_M = 0$. $current_d = 0$
5: for all internal nodes $n_i \in H$ in top down fashion do
6:     if $n_i$.split == $true$ then
7:         for all child $\in n_i$ do
8:             if child.split == $true$ then
9:                 assign each query in child to separate partition in round-robin fashion
10:            else
11:                push child for processing
12:            end if
13:        end for
14:    else
15:        if $n_i$.projectedcost <($B_{exp}$ - $B[current_d]$) then
16:            Add the queries belonging to node in the current partition
17:            $q_i \in n_i$, add $q_i$ to $D[current_d]$
18:            $B[current_d]$ += $n_i$.projectedcost
19:        else
20:            access all children of $n_i$ one by one
21:            Repeat step 7 for each children
22:        end if
23:    end if
24:    if $B[current_b]$ >= $B_{exp}$ then
25:        $current_b = current_b + 1$
26:    end if
27: end for
```

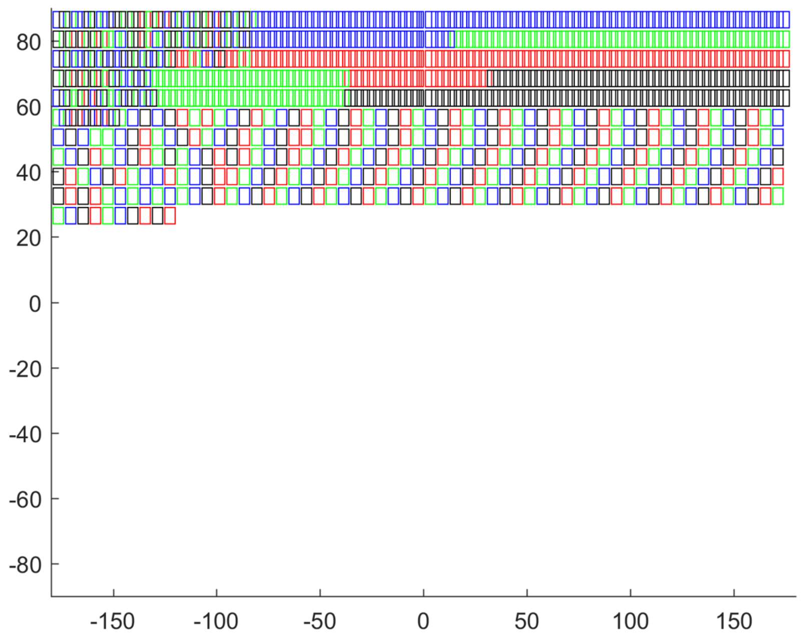

Figure 4.15: Results for query workload of 1200 queries using MaxCC-S with number of partitions M = 4.

lowing are the four partitions created. Refer figure 4.12 Each partition is marked in different colors. Less than 10 percent overlaps marks nodes for separation strategy.

4.5.4 Conclusion

The approach partitions the query workload into load-balanced partitions. These partitions take similar time to process. The read bitmap cost is low. The read bitmap cost is higher than first approach. But, the combine bitmap cost is optimized in this approach. The overall results for this approach performs better than first approach when there are less overlaps. Refer experiments in Chapter 5 for more details.

Chapter 5

EXPERIMENTAL EVALUATION

In this section, I evaluate the performance of the system.

5.1 Experimental setup and dataset

Experiments are conducted on a machine running Ubuntu 16.04 operating system, 8 vCPU, 16 GB of memory and 160 GB Disk. The instance contains Java version 8.

I have conducted tests on the following datasets:

1. **Uniform dataset**: The first set of experiments are performed on uniform distributed dataset containing 100M data points spread across range (-180, -90) to (180, 90). The cSHB index structure created for this data is used for evaluating the performance of the workload partitioning.

2. **Real world dataset**: The second set of experiments are performed on non-uniform distributed dataset containing 2.7M data points from gowalla dataset[19] spread across range (-180, -90) to (180, 90). The cSHB index structure created for this data is used for evaluating the performance of the workload partitioning. Different query workloads were generated with queries corresponding to the US and the Europe regions.

5.2 Approaches

The workload is partitioned using 3 algorithms:

1. Max Containment Clustering (MaxCC)

2. Max Containment Clustering with Separation (MaxCC-S)

3. Random Clustering Approach

The first and second approach is discussed in section 4. Both these approaches produce a partition set P_{app1} and P_{app2}. The Random clustering approach creates the partitions based on the random queries. For example- If $Q = [q_1, q_2, q_3, q_4, q_5, q_6]$ and we need to create number of partitions (M)= 2 then it randomly picks the queries for assigning it to a partition in round-robin fashion.

Random Clustering output for M=2:

Partition Set $P = [p_1, p_2]$

partition $p_1 = [q_6, q_1, q_5]$

partition $p_2 = [q_2, q_3, q_4]$

To test multiple random partitions, I have created 5 to 10 random partitions (partition set) for same value of M. This is to test the behavior of different random clustering outputs. In order to compare these 5 random partition set with MaxCC and MaxCC-S, I have sorted each partition set from 5 of partition set using ascending order of performance of their individual partitions. I have taken average of all 5 and finally we get one partition set to compare with other 2 approaches.

Example: 5 partition set M=2

$P1 = [p_{11}, p_{12}]$

$P2 = [p_{21}, p_{22}]$

$P3 = [p_{31}, p_{32}]$

$P4 = [p_{41}, p_{42}]$

$P5 = [p_{51}, p_{52}]$

Sort partitions in Partition set by their performance

$P1 = [p_{12}, p_{11}]$

$P2 = [p_{22}, p_{21}]$

$P3 = [p_{31}, p_{32}]$

$P4 = [p_{42}, p_{41}]$

$P5 = [p_{51}, p_{52}]$

Now, take the average of each corresponding first and second partition in all 5 partitions to get one partiton set $P_{random} = [p_1, p_2]$. This will be compared with performance of approaches 1 and 2.

5.3 Performance Evaluation

The tests are conducted to check the performance of approaches on both datasets. The test scenarios include varying number of partitions (M). Testing scenarios are based on the degree of overlaps-

Test 1: No overlaps

Test 2: 0 to 10% overlaps

Test 3: 20 to 40% overlaps

Test 4: 40 to 80% overlaps

Test 5: 0 to 80% overlaps

Test 6: 0 to 40% overlaps

Currently M is considered as 2, 4, 6, 8 for each test scenario. Multiple runs are performed for each test scenario and variance is checked in order to report the unbiased behavior of each test scenario.

The query workload Q for different test scenarios should be partitioned into M-partitions that are load-balanced. These query partitions created are tested individually for their query processing time. The conclusion described in each approach must be reflected in the results. The major performance gain should be seen in the Disk IO and CPU operations for bitmaps. These can be observed using the read bitmap and combine bitmap operation processing time. The performance of both techniques are compared with random partitioning. The query workload generated has varying degree of overlaps. Based on these overlaps, the partitions created are different for both techniques.

Example for test scenario that contains workload with 0 to 80% overlaps: The figure 5.1 shows the performance of 3 approaches for each partition for given M partitions.

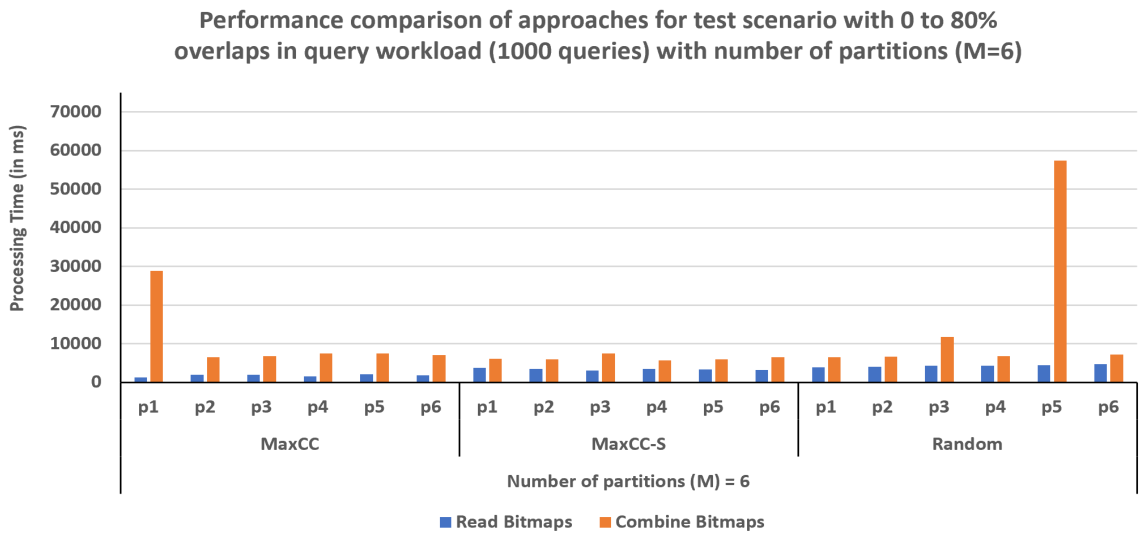

Figure 5.1: Performance comparison of approaches for each partition for $M = 6$

Observations: The figure 5.1 contains groupings based on approaches and the partition numbers for $M = 6$. MaxCC shows low disk access for all partitions. Check the blue bar in the figure 5.1. MaxCC-S shows optimized combine bitmap operations as compared to MaxCC. Random approach performs worst as it has high block reads that increases cost of read bitmaps. The results are as per expectations of each approach for $M = 6$. The detailed analysis in section 5.4 explains such test scenarios for varying values of $M = [2, 4, 6, 8]$.

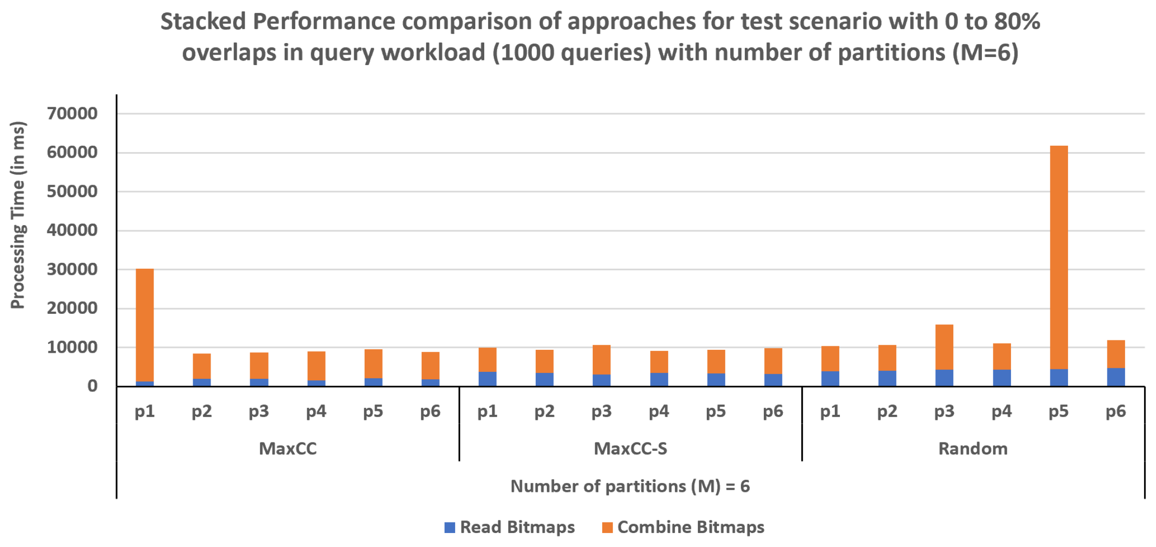

Figure 5.2: Stacked performance comparison of approaches for each partition for $M = 6$

The figure 5.2 shows the stacked representation of both performance parameters (read and combine bitmaps). It provides a clear understanding about the effect of unbalanced partition in random partitioning can lead to increase in processing time of overall query workload. The MaxCC saves time in reading the bitmaps but are not able to completely optimize the bitmap operations. The MaxCC-S can be helpful in such scenarios of varying degree of overlaps from 0 to 80%.

		Test 1		Test 2		Test 3		Test 4		Test 5	
M	**Exp.**	**R**	**C**	**R**	**C**	**R**	**C**	**R**	**C**	**R**	**C**
2	**<50**	80	73	38	43	37	49	33	49	39	41
4	**<25**	31	12	13	20	21	29	19	23	22	23
6	**<17**	21	9	10	13	12	17	12	14	13	14
8	**<12**	16	9	8	10	10	12	9	11	12	10

Table 5.1: Average processing time (in percentage) for varying number of partitions(M). Percentage is calculated based on the processing time of query workload without partitions. (R:Read Bitmaps, C: Combine Bitmaps, Exp: Expected Percentage)

Refer Table 5.1 for the performance comparison across varying number of partitions M. The average processing time of partitions is compared with processing time of query workload without partitioning. R denotes Read bitmap percentage, C denotes Combine bitmap percentage. Example- For M=2, The expected (Exp) value in table <50% which means the processing time should take less than 50% after partitioning. For test 1 with no overlaps both R and C has poor performance. But for other tests (2,3,4,5), R and C values show promising results. For M = 4, the performance expected is less than 25% of the processing time of entire query workload without partitions. For all test scenarios M = 4 performs best. Similarly, for M = [6,8] shows performance improvement more in combine bitmap operations.

Next section 5.4 provides results and explanation for different test scenarios and performance of each approach.

5.4 Detailed Analysis

5.4.1 Dataset 1: Uniform dataset

I have created query workloads with varying size of overlap percent (0 to 80%) and query range size = 1% to 1.5%. Query Workload contains total queries from 1000 to 2000. Using MaxCC and MaxCC-S, the query workload is partitioned. Number of partitions(M) has varying values 2, 4, 6, 8. The partitioning approaches are tested for all these number of partitions. The results are organized into two types of visualization. The detailed view is displayed using the bar chart. The overview of varying number of partitions M for test scenario is visualized using bar chart grouped by number of partitions.

Test 1.1: Test scenario with no overlaps

Description: In this test scenario, the range query workload consists of set of queries that does not overlap with each other. The queries are very closely located to each other but does not intersect.

Results:

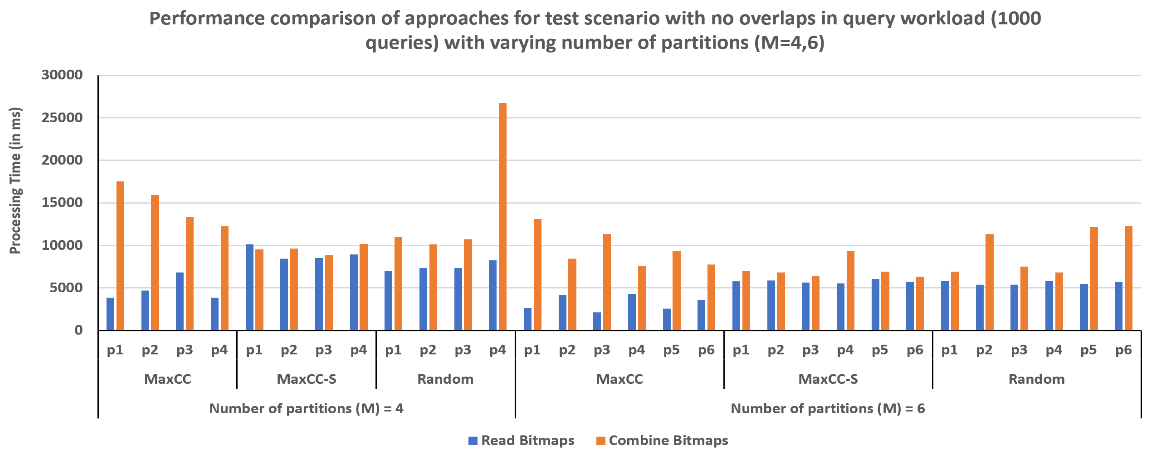

Figure 5.3: Performance comparison of approaches for test scenario with no overlaps in query workload (1000 queries) with varying number of partitions $M = [4, 6]$

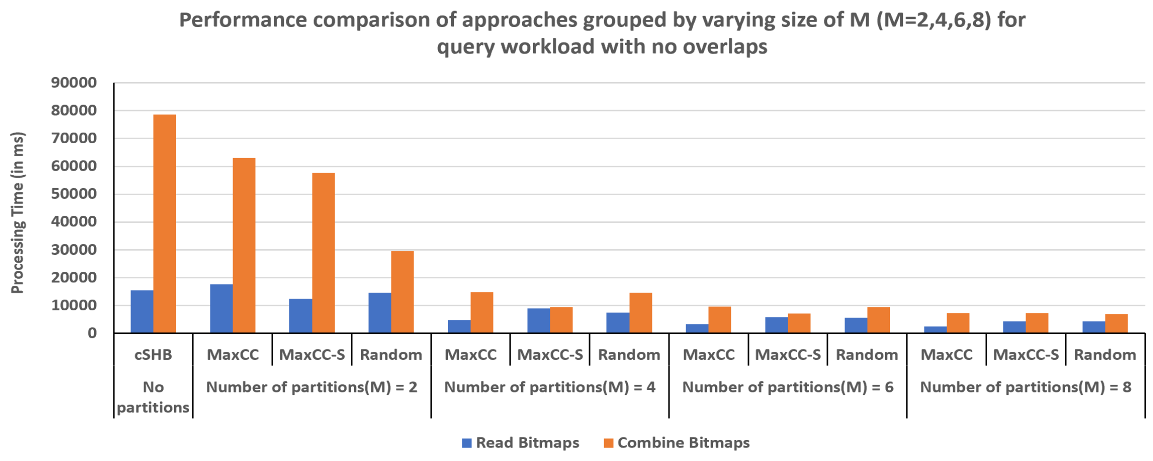

Figure 5.4: Performance comparison of approaches grouped by varying size of $M = [2, 4, 6, 8]$ for query workload with no overlaps

Observation for Test 1.1:

1. Read Bitmaps: Refer figure 5.3. The less number of block reads lowers the disk access. Thus all partitions in MaxCC will have less read bitmap cost(shown in blue color). For MaxCC-S, the cost for read bitmap is a bit higher than the MaxCC.

   Refer figure 5.4. The read bitmap performance is shown in blue color. Read bitmap cost is minimum(as expected) for MaxCC for $M = [4, 6, 8]$.

2. Combine Bitmaps: Refer figure 5.3, the combine bitmap operations are optimized in second approach. It performs best and reduces the CPU operations. MaxCC has skewed results for few of the partitions. All the results are as expected.

   Refer figure 5.4. the combine bitmap is shown in orange color. After observing the figure 5.4 for all partitions, the trench or drop is seen near MaxCC-S. This suggests that there is a drop in combine bitmap cost (as per expectation). When number of partitions are 4 or more, then the approaches perform best for this test scenario.

3. Performance Gain: Refer 5.4 Performance gain increases for $M = [4, 6, 8]$ if the values are compared with no partitions. For $M = 2$ the read bitmaps cost has very less up to 5% gains. But for M=4, the performance is improved and cost gets reduced to almost 30% of the read cost without partitions. For combine bitmaps the cost gets reduced to almost 12-15% of the combine bitmaps cost of the query workload without partitions. Similar trend can be observed for rest of the number of partitions. The performance of the system improves if there are more than 150 queries in each partition. Thus the performance is best for $M = [4, 6]$ and good for $M = 8$.

Test 1.2: Test scenario with 0 to 10% overlaps

Description: In this test scenario, the range query workload consists of set of queries that has overlap degree between 0 and 10%. The overlap percent depends on the overlapping region of the query. Different number of partitions are created using approaches mentioned earlier and their performance is compared in the next section.

Results:

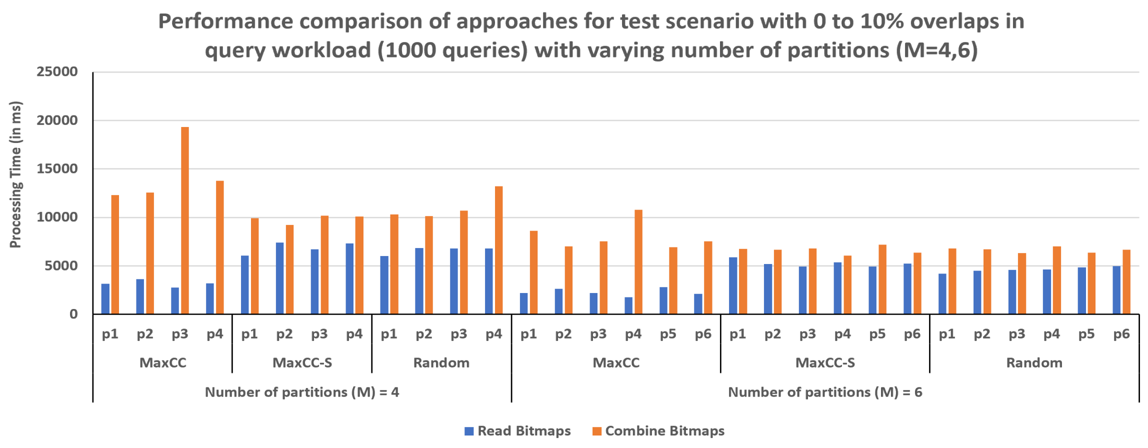

Figure 5.5: Performance comparison of approaches for test scenario with 0 to 10% overlaps in query workload (1000 queries) with varying number of partitions $M = [4, 6]$

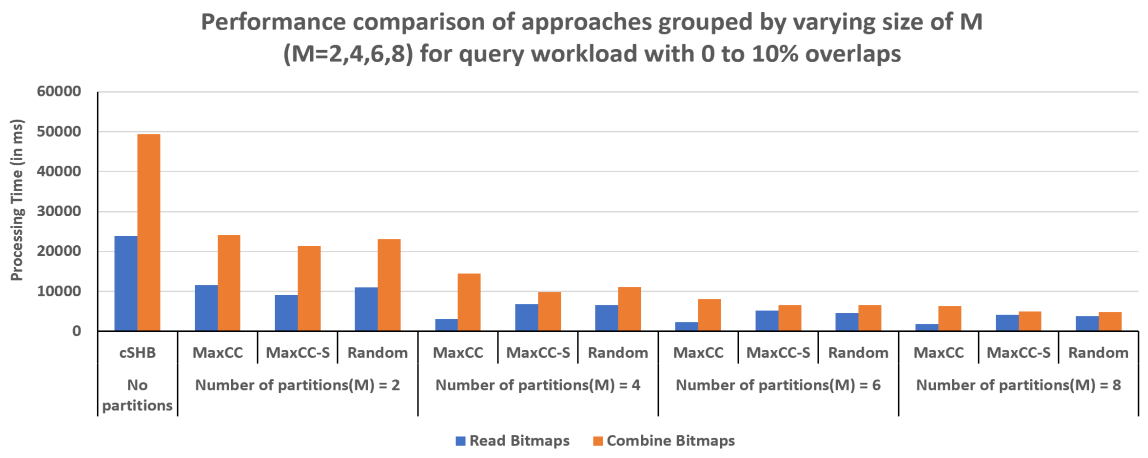

Figure 5.6: Performance comparison of approaches grouped by varying size of $M = [2, 4, 6, 8]$ for query workload with 0 to 10% overlaps

Observation for Test 1.2:

1. Read Bitmaps: Refer figure 5.5. The less number of block reads lowers the disk access. Thus all partitions in MaxCC (Max Containment Clustering) will have

less read bitmap cost(shown in blue color). For MaxCC-S, the cost for read bitmap is a bit higher than the MaxCC.

Refer figure 5.6. The read bitmap performance is shown in blue color. Read bitmap cost is minimum for MaxCC for $M = [4, 6, 8]$. For $M = 2$, the MaxCC has almost similar time as MaxCC-S for reading the bitmaps.

2. Combine Bitmaps: Refer figure 5.5, the combine bitmap operations are optimized in MaxCC-S approach. It performs best and reduces the CPU operations. For combine bitmap performance, MaxCC has skewed results for few of the partitions. Example- Check MaxCC, M=4, p3 partition. MaxCC-S handles this by using separation criteria(<10% overlap) and performance gain can be seen in MaxCC-S for M=2. Similary, for rest of the values of M, MaxCC-S performs better. All the results are as expected.

   Refer figure 5.6. the combine bitmap is in orange color. After observing the figure 5.4 for all partitions, the trench or drop is seen near MaxCC-S. This suggests that there is a drop in combine bitmap cost (as per expectation). When number of partitions are 2 and 4, then the approaches perform best for this test scenario.

3. Performance Gain: Refer figure 5.6 Performance gain increases for $M = [2, 4, 6, 8]$ if the values are compared with no partitions. For $M = 2$ the read bitmaps cost has reduced to 40% of read cost without partitions. For M=4, the performance is improved and cost gets reduced to almost 13% of the read cost without partitions. For combine bitmaps the cost gets reduced to almost 20-30% of the combine bitmaps cost of the query workload without partitions. Similar trend can be observed for rest of the number of partitions. The performance of the system improves if there are more than 100 queries in each partition. Thus the

performance is best for $M = [4, 6, 8]$.

Test 1.3: Test scenario with 20 to 40% overlaps

Description: In this test scenario, the range query workload consists of set of queries that has overlap degree between 20 and 40%. The overlap percent depends on the overlapping region of the query.

Results:

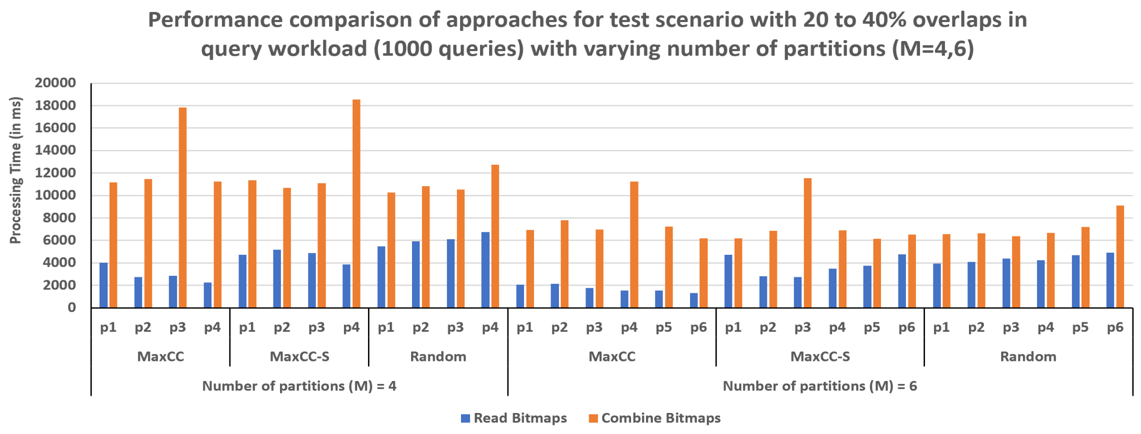

Figure 5.7: Performance comparison of approaches for test scenario with 20 to 40% overlaps in query workload (1000 queries) with varying number of partitions $M = [4, 6]$

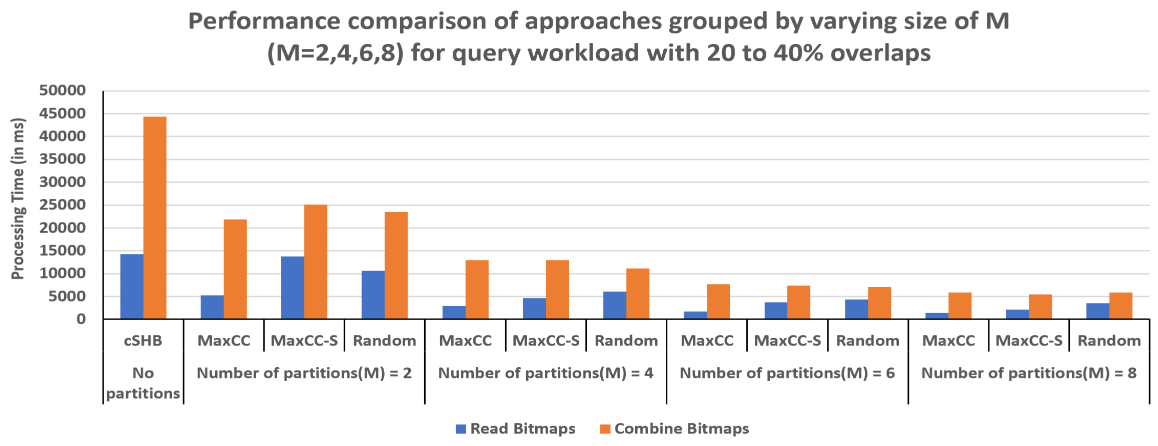

Figure 5.8: Performance comparison of approaches grouped by varying size of $M = [2, 4, 6, 8]$ for query workload with 20 to 40% overlaps

Observation for Test 1.3:

1. Read Bitmaps: Refer figure 5.7. The less number of block reads lowers the disk access. Thus all partitions in MaxCC (Max Containment Clustering) will have less read bitmap cost(shown in blue color). For MaxCC-S, the cost for read bitmap is a bit higher than the MaxCC.

   Refer figure 5.8. The read bitmap performance is shown in blue color. Read bitmap cost is minimum for MaxCC for $M = [2, 4, 6, 8]$.

2. Combine Bitmaps: Refer figure 5.7, the combine bitmap operations are optimized in MaxCC-S approach. In this particular scenario with 20 to 40% overlaps, the separation strategy does not find queries to separately cluster. As a result, it acts almost similar to MaxCC and combines the nearby queries.

   Refer figure 5.8, the combine bitmap is shown in orange color. The overlaps above 20% for all queries causes MaxCC-S to act similar to MaxCC. Due to this, the MaxCC and MaxCC-S results are similar when number of partitions are 4, 6, 8.

3. Performance Gain: Refer figure 5.8 Performance gain increases for $M = [2, 4, 6, 8]$ if the values are compared with no partitions. For $M = 2$ the read bitmaps cost has reduced to 37% of the read cost without partitions in MaxCC approach. But for M=4, the performance is improved and cost gets reduced to almost 21% of the read cost without partitions. For combine bitmaps, the cost gets reduced to almost 30% of the combine bitmaps cost of the query workload without partitions. Similar trend can be observed for rest of the number of partitions. The performance of the system improves if there are more than 100 queries in each partition. Thus the performance is best for $M = [2, 4, 6]$ and good for $M = [8]$.

Test 1.4: Test scenario with 0 to 80% overlaps

Description: In this test scenario, the range query workload consists of set of queries that has overlap degree between 0 and 80%. The overlap percent depends on the overlapping region of the query. This overlapping scenario with varying overlap degree shows the advantage of both approaches clearly.

Results:

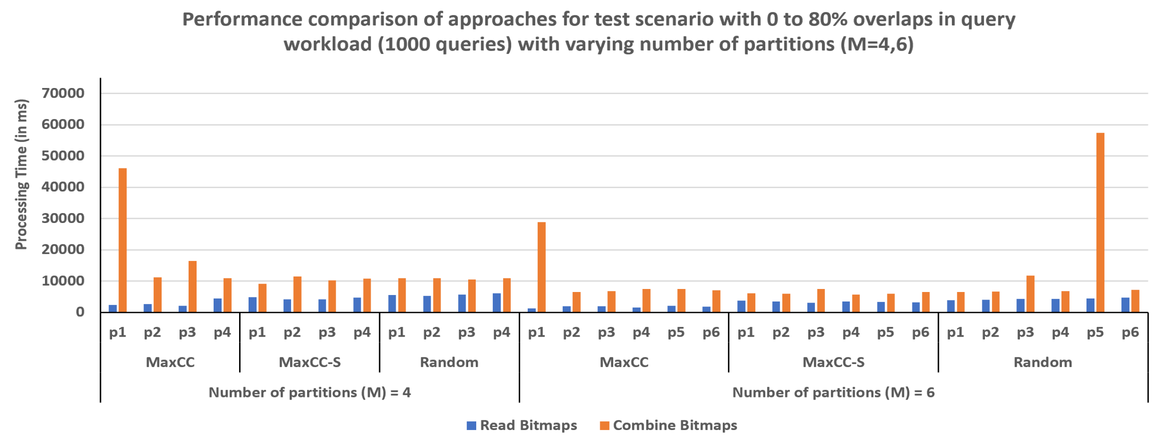

Figure 5.9: Performance comparison of approaches for test scenario with 0 to 80% overlaps in query workload (1000 queries) with varying number of partitions $M = [4, 6]$

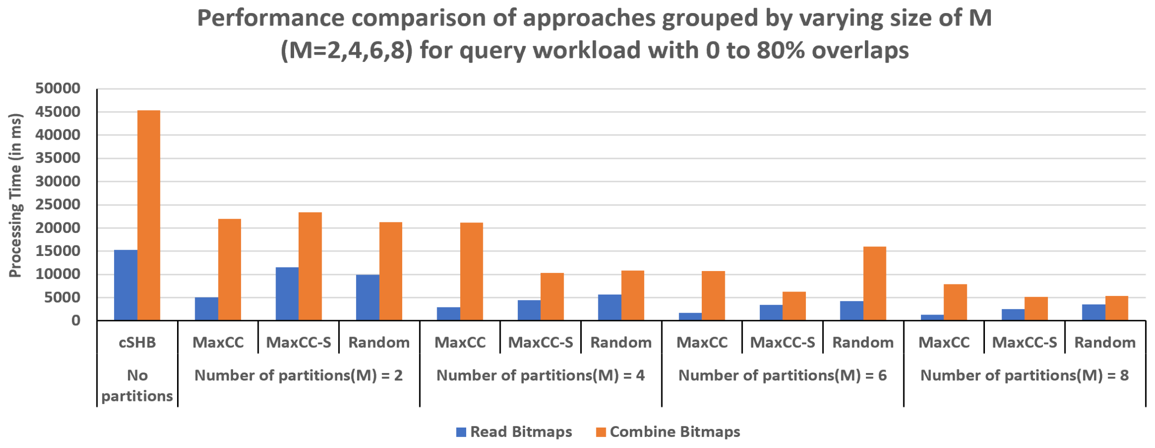

Figure 5.10: Performance comparison of approaches grouped by varying size of $M = [2, 4, 6, 8]$ for query workload with 0 to 80% overlaps

Observation for Test 1.4:

1. Read Bitmaps: Refer figure 5.9. The less number of block reads lowers the disk access. Thus all partitions in MaxCC (Max Containment Clustering) will have

less read bitmap cost(shown in blue color). For MaxCC-S, the cost for read bitmap is a bit higher than the MaxCC(as per expectation).

Refer figure 5.10. The read bitmap performance is shown in blue color. Read bitmap cost is very low for MaxCC for $M = [2, 4, 6, 8]$.

2. Combine Bitmaps: Refer figure 5.9, the combine bitmap operations are optimized in second approach. In this particular scenario with 0 to 80% overlaps, the separation approach works perfect and finds the ideal clustering for queries. As a result, in figure 5.9, consistency can be observed for all the partitions for MaxCC-S (check orange color bars).

   Refer figure 5.10. the combine bitmap is in orange color. The results clearly show the trench or drop for MaxCC-S for $M = [4, 6, 8]$. Therefore, the performance of combine bitmaps in MaxCC-S is as per expectation.

3. Performance Gain: Refer figure 5.10 Performance gain increases for $M = [2, 4, 6, 8]$ if the values are compared with no partitions. For $M = 2$ the read bitmaps cost has reduced to 30% of read cost without partitions. For M=4, the performance is improved and cost gets reduced to almost 19% of the read cost without partitions. For combine bitmaps the cost gets reduced to almost 50%(in MaxCC) and 25%(in MaxCC-S) of the combine bitmaps cost of the query workload without partitions. Similar trend can be observed for rest of the number of partitions. The performance of the system improves if there are more than 100 queries in each partition. Thus the performance is best for $M = [4, 6, 8]$ for MaxCC-S.

Test 1.5: Test scenario with 0 to 40% overlaps

Description: In this test scenario, the range query workload consists of set of queries that has overlap degree between 0 and 40%.

Results:

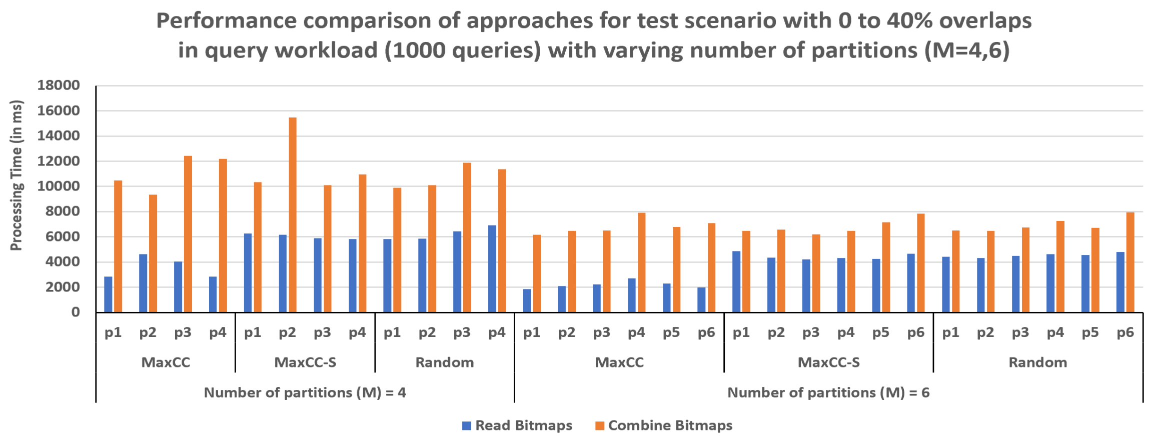

Figure 5.11: Performance comparison of approaches for test scenario with 0 to 40% overlaps in query workload (1000 queries) with varying number of partitions $M = [4, 6]$

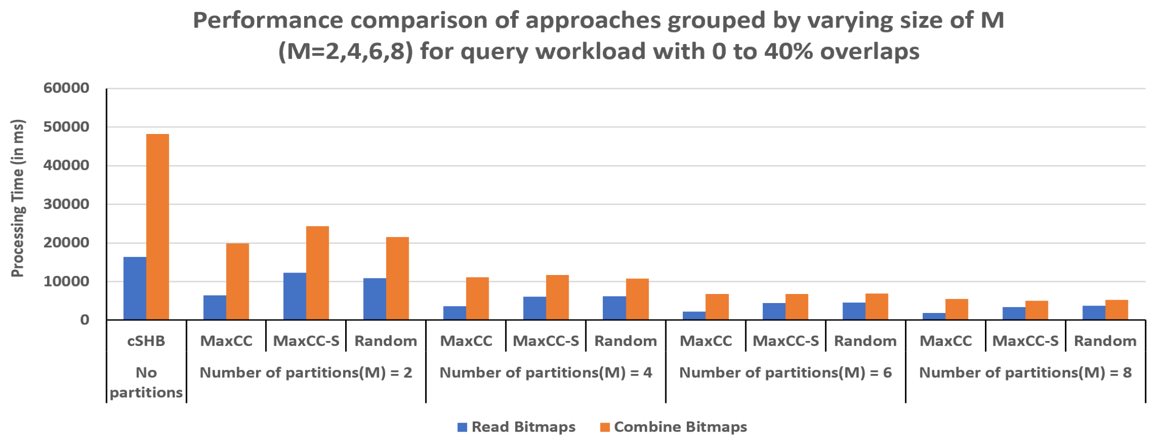

Figure 5.12: Performance comparison of approaches grouped by varying size of $M = [2, 4, 6, 8]$ for query workload with 0 to 40% overlaps

Observation for Test 1.5:

1. Read Bitmaps: Refer figure 5.11. The overlaps take advantage of grouping similar queries in same cluster. Thus, less number of blocks are read that

lowers the disk access. As a result, all partitions in MaxCC (Max Containment Clustering) will have less read bitmap cost(shown in blue color). For MaxCC-S, the cost for read bitmap is a bit higher than the MaxCC(as per expectation).

   Refer figure 5.12. The read bitmap performance is shown in blue color. Read bitmap cost is very low for MaxCC for all values of M

2. Combine Bitmaps: Refer figure 5.11, the combine bitmap operations are optimized in second approach. In this particular scenario with 0 to 40% overlaps, the separation approach may or may not find the ideal clustering for queries as separation criteria(<10%). As a result, in figure 5.11, MaxCC-S has combine bitmap cost similar to MaxCC (check orange color bars for M=6).

   Refer figure 5.12. the combine bitmap is shown in orange color. The results shows the almost similar values for MaxCC and MaxCC-S for $M = [4, 6, 8]$.

3. Performance Gain: Refer figure 5.10 Performance gain increases for $M = [2, 4, 6, 8]$ if the values are compared with no partitions. For $M = 2$ the read bitmaps cost has reduced to 40% of read cost without partitions. For M=4, the performance is improved and cost gets reduced to almost 22% of the read cost without partitions. For combine bitmaps the cost gets reduced to almost 23% of the combine bitmaps cost of the query workload without partitions. Similar trend can be observed for rest of the number of partitions. The performance of the system improves if there are more than 100 queries in each partition. Thus the performance is best for $M = [4, 6, 8]$.

5.4.2 Dataset 2: Real world dataset

I have created query workloads with varying size of overlap percent (0 to 80%) and query range size = 0.8% to 1%. The queries in query workload are created for US and Europe region by using the latitude and longitude values of these regions. Query Workload contains total queries from 100 to 500. Using MaxCC and MaxCC-S, the query workload is partitioned. Random partitioning for same workloads is performed to compare the approaches. Number of partitions(M) has varying values 2, 4, 6, 8. The partitioning approaches are tested for all these number of partitions. The results are organized into two types of visualization - the detailed view (bar chart) and the overview of varying number of partitions M (bar chart).

Test 2.1: Test scenario with 40 to 80% overlaps

Description: In this test scenario, the range query workload consists of set of queries that has overlap degree between 40 and 80%.

Results:

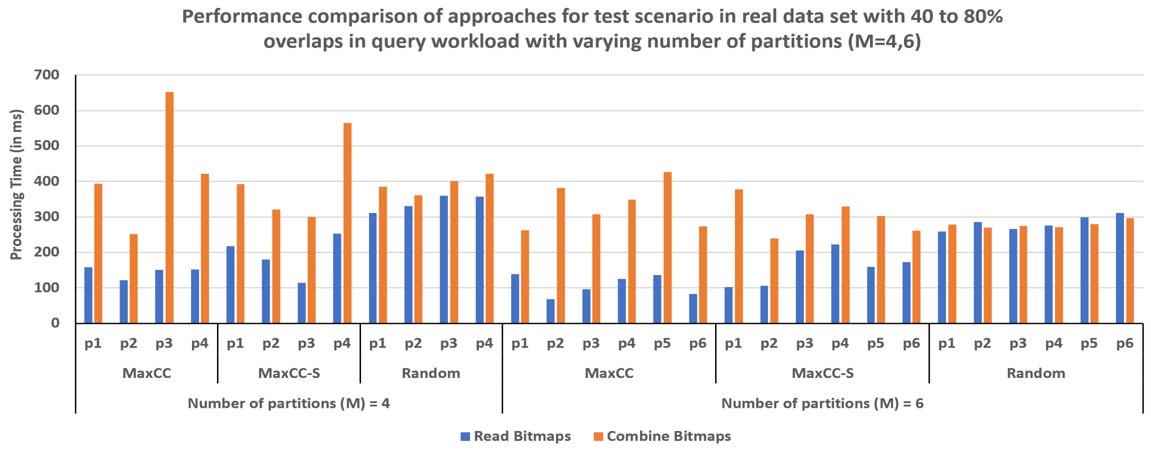

Figure 5.13: Performance comparison of approaches for test scenario in real dataset with 40 to 80% overlaps in query workload with varying number of partitions $M = [4, 6]$

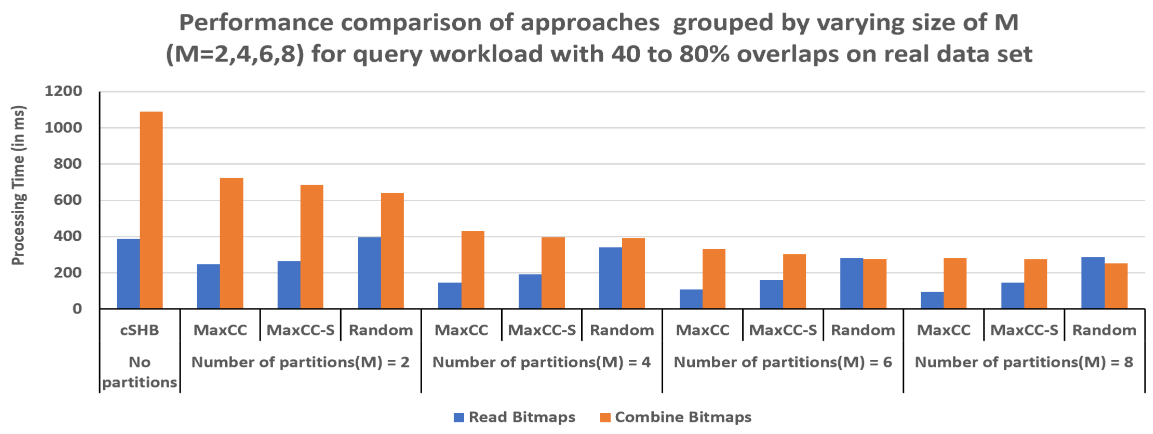

Figure 5.14: Performance comparison of approaches grouped by varying size of $M = [2, 4, 6, 8]$ for query workload with 40 to 80% overlaps on real dataset

Observation for Test 2.1:

1. Read Bitmaps: Refer figure 5.13. The overlaps take advantage of grouping similar queries in same cluster. Thus, less number of blocks are read that lowers the disk access. As a result, all partitions in MaxCC (Max Containment Clustering) will have less read bitmap cost(shown in blue color). For MaxCC-S, the cost for read bitmap is a bit higher than the MaxCC(as per expectation).

   Refer figure 5.14. The read bitmap performance is shown in blue color. Read bitmap cost is very low for MaxCC for all values of M

2. Combine Bitmaps: Refer figure 5.13, the combine bitmap operations are optimized in second approach. In this particular scenario with 40 to 80% overlaps, the separation approach may or may not find the good clustering for queries as separation criteria(<10%). As a result, in figure 5.13, MaxCC-S has combine bitmap cost similar to MaxCC (check orange color bars for M=6).

   Refer figure 5.14. the combine bitmap is shown in orange color. The results shows the almost similar values for MaxCC and MaxCC-S for $M = [4, 6, 8]$. Random partitioning has higher values than both proposed approaches.

3. Performance Gain: Refer figure 5.14 Performance gain increases for $M = [2, 4, 6, 8]$ if the values are compared with no partitions. For $M = 2$ the read bitmaps cost has reduced to 60% of read cost without partitions. For M=4, the performance is improved and cost gets reduced to almost 35% of the read cost without partitions. For combine bitmaps the cost gets reduced to almost 35% of the combine bitmaps cost of the query workload without partitions. Similar trend can be observed for rest of the number of partitions. The performance of the system improves if there are more than 75 queries in each partition. The performance is best for $M = [4, 6]$.

Test 2.2: Test scenario with 0 to 40% overlaps

Description: In this test scenario, the range query workload consists of set of queries that has overlap degree between 0 and 40%.

Results:

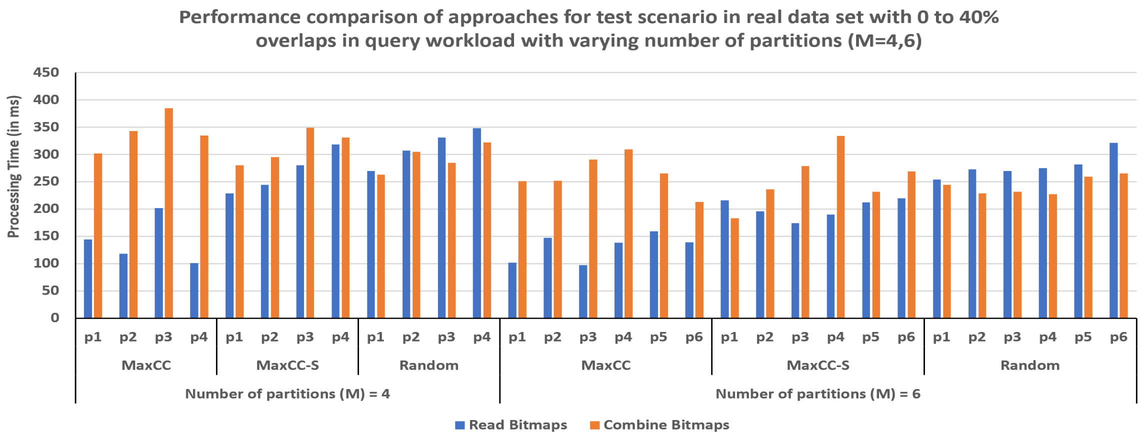

Figure 5.15: Performance comparison of approaches for test scenario in real dataset with 0 to 40% overlaps in query workload with varying number of partitions $M = [4, 6]$

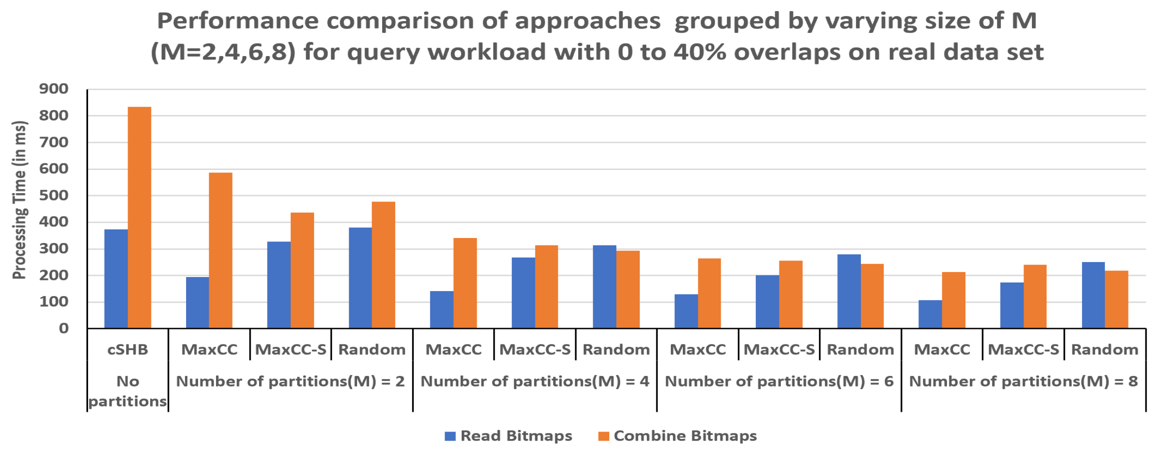

Figure 5.16: Performance comparison of approaches grouped by varying size of $M = [2, 4, 6, 8]$ for query workload with 0 to 40% overlaps on real dataset

Observation for Test 2.2:

1. Read Bitmaps: Refer figure 5.15. All the partitions in MaxCC (Max Containment Clustering) will have less read bitmap cost(shown in blue color). For MaxCC-S, the cost for read bitmap is a bit higher than the MaxCC(as per expectation).

   Refer figure 5.16. The read bitmap performance is shown in blue color. Read bitmap cost is very low for MaxCC for all values of M

2. Combine Bitmaps: Refer figure 5.15, the combine bitmap operations are optimized in second approach. In this particular scenario with 0 to 40% overlaps, the separation approach may find the good clustering for queries as separation criteria(<10%). As a result, in figure 5.15, MaxCC-S has combine bitmap cost is lower than MaxCC. (check orange color bars for M=4).

   Refer figure 5.16. the combine bitmap is shown in orange color. The results shows the almost similar values for MaxCC and MaxCC-S for $M = [4, 6, 8]$.

Random partitioning has higher values than both proposed approaches.

3. Performance Gain: Refer figure 5.16, performance gain increases for $M = [2, 4, 6, 8]$ if the values are compared with no partitions. For $M = 2$, the read bitmaps cost has reduced to 60% of read cost without partitions. For M=4, the performance is improved and cost gets reduced to almost 35-40% of the read cost without partitions. For combine bitmaps the cost gets reduced to almost 40% of the combine bitmaps cost of the query workload without partitions. Similar trend can be observed for rest of the number of partitions. The performance of the system improves if there are more than 75 queries in each partition. Thus the performance is best for $M = [4]$ and good for $M = [6]$.

Test 2.3: Test scenario with no overlaps

Description: In this test scenario, the range query workload consists of set of queries that has no overlaps.

Results:

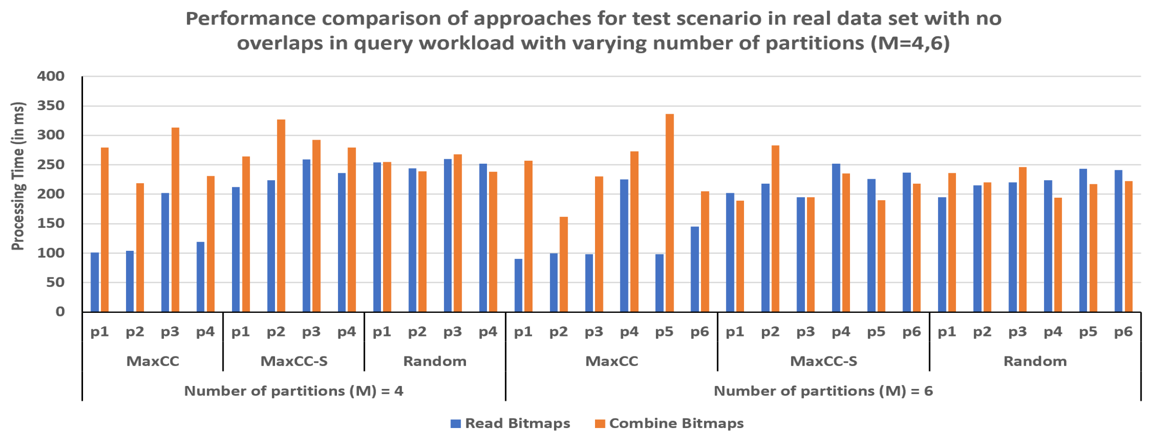

Figure 5.17: Performance comparison of approaches for test scenario in real dataset with no overlaps in query workload with varying number of partitions $M = [4, 6]$

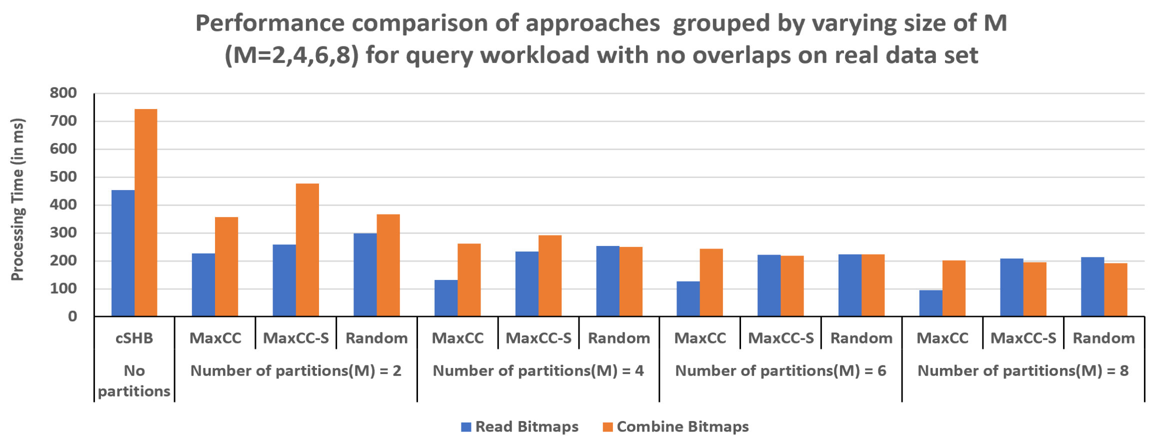

Figure 5.18: Performance comparison of approaches grouped by varying size of $M = [2, 4, 6, 8]$ for query workload with no overlaps on real dataset

Observation for Test 2.3:

1. Read Bitmaps: Refer figure 5.17. All the partitions in MaxCC (Max Containment Clustering) will have less read bitmap cost(shown in blue color). For MaxCC-S, the cost for read bitmap is a bit higher than the MaxCC(as per expectation). Random has a very high read bitmap cost.

   Refer figure 5.18. The read bitmap performance is denoted by circle as a marker(shown in blue color). Read bitmap cost is very low for MaxCC for all values of M.

2. Combine Bitmaps: Refer figure 5.17, the combine bitmap operations are optimized in second approach. In this particular scenario with 0% overlaps, the separation approach finds the good clustering for queries as separation criteria(<10%). As a result, in figure 5.15, MaxCC-S has combine bitmap cost consistent than MaxCC. (check orange color bars for M=[4,6]).

   Refer figure 5.18. the combine bitmap is shown using triangle marker(orange color). The results shows the almost similar values for MaxCC and MaxCC-S for $M = [4, 6, 8]$. Random partitioning has higher values than both proposed approaches.

3. Performance Gain: Refer figure 5.18, performance gain increases for $M = [2, 4, 6, 8]$ if the values are compared with no partitions. For $M = 2$, the read bitmaps cost has reduced to 50% of read cost without partitions. For M=4, the performance is improved and cost in MaxCC gets reduced to almost 25% of the read cost without partitions. For combine bitmaps the cost gets reduced to almost 40% of the combine bitmaps cost of the query workload without partitions. Similar trend can be observed for rest of the number of partitions. The performance of the system improves if there are more than 75 queries in each partition. Thus the performance is best for $M = [2, 4]$. The performance improvement drops if the number of partitions are increased.

5.5 Results Overview

1. **Low bitmap access**: *MaxCC- Max Containment Clustering with load-balanced partitions* groups nearby queries into same clusters. For all test scenarios, the disk access is low for MaxCC. It performs best in case of more overlapping degree. The queries can find the ideal clustering scenario that access neighboring nodes. Note that in cSHB indexes[23], these neighboring nodes are stored in same block. Thus, it affects the disk access cost and provides performance boost in read bitmaps shown in the detailed analysis section.

2. **Optimized bitmap operations**: *MaxCC-S- Max Containment Clustering with Separation* uses separation criteria for partitioning the queries in the workload. This criteria used for above experiments is less than 10% overlap. If the

overlap is less than 10% then it will work great by optimizing the bitmap operation for the blocks accessed. This also provides best results when a query workload with 0 to 80% overlaps is provided. It can take advantage of this varying overlapping region to find ideal clusters that helps in optimizing the CPU operations.

3. **Varying number of partitions**: The number of partitions $M = 2$ is very low for query workload of 1000 queries. For such low number of partitions, the partitioning algorithm for MaxCC-S cannot create ideal clusters. As a result, separation of queries is not possible and can reduce the performance. The number of partitions M should be selected according to the query workload count to get best results. The number of queries in the partitions also affect the performance. At least 75-125 queries in each partition shows the performance improvement. The results discussed in section 5.4 shows good results when number of partitions M are 4,6,8.

4. **Effects of Random Partitioning**: The random partition behaves in a random manner. For some partitions it finds best results but for others it gets worst performance. The random partitioning has a lot of chances of causing resource overloading bottlenecks in parallel execution.

Chapter 6

CONCLUSION

The two approaches proposed in the thesis create partitions for processing the query workload in parallel. The load-balanced partitions help in reducing the resource overloading bottlenecks that are caused by unbalanced partitions.

The first approach, Max Containment Clustering, lowers the blocks read using the cSHB indexes. This produces very low disk accesses that provide a performance boost in the read bitmap cost internally in the cSHB. The neighboring queries are clustered together which helps in lowering block reads using the block-based organization of the cSHB index hierarchy. The combine bitmap operations are low but not best as compared to the second approach.

The second approach, Containment Clustering with Separation Strategy, optimizes the cost for bitmap operations. The strategy splits the queries in separate partition to lower the bitmap computations. This produces the performance gain in the combine bitmap operations. The read bitmap cost is a bit higher than approach 1 due to the separation of neighboring queries. The overall performance of the approach is best for query workloads with varying degree of overlaps.

Partitions generated and results obtained are as expected from both approaches.

In future, these strategies can be adapted to other index structures to increase their performance by partitioning query workloads as per the expectation. The sample query workloads can be tested on other indexing structures. Using these results, approaches discussed can be used directly or can be configured according to the results.

REFERENCES

[1] "Clustering", URL https://www-users.cs.umn.edu/~kumar001/dmbook/ch8.pdf (2015).

[2] "R-tree", URL https://www.ibm.com/support/knowledgecenter/en/SSGU8G_11.50.0/com.ibm.rtree.doc/sii-overview-27706.htm (2015).

[3] "R* tree", URL https://en.wikipedia.org/wiki/R*_tree (2015).

[4] "R*-tree", URL https://github.com/davidmoten/rtree (2015).

[5] Aji, A., F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang and J. Saltz, "Hadoop gis: a high performance spatial data warehousing system over mapreduce", Proceedings of the VLDB Endowment **6**, 11, 1009–1020 (2013).

[6] Aljawarneh, I. M., P. Bellavista, A. Corradi, R. Montanari, L. Foschini and A. Zanotti, "Efficient spark-based framework for big geospatial data query processing and analysis", in "Computers and Communications (ISCC), 2017 IEEE Symposium on", pp. 851–856 (IEEE, 2017).

[7] Aly, A. M., A. R. Mahmood, M. S. Hassan, W. G. Aref, M. Ouzzani, H. Elmeleegy and T. Qadah, "Aqwa: adaptive query workload aware partitioning of big spatial data", Proceedings of the VLDB Endowment **8**, 13, 2062–2073 (2015).

[8] Bandyopadhyay, S. and E. J. Coyle, "An energy efficient hierarchical clustering algorithm for wireless sensor networks", in "INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies", vol. 3, pp. 1713–1723 (IEEE, 2003).

[9] Beckmann, N., H.-P. Kriegel, R. Schneider and B. Seeger, "The r*-tree: an efficient and robust access method for points and rectangles", in "Acm Sigmod Record", vol. 19, pp. 322–331 (Acm, 1990).

[10] Brinkhoff, T., H.-P. Kriegel and B. Seeger, "Parallel processing of spatial joins using r-trees", in "Data Engineering, 1996. Proceedings of the Twelfth International Conference on", pp. 258–265 (IEEE, 1996).

[11] Dean, J. and S. Ghemawat, "Mapreduce: simplified data processing on large clusters", Communications of the ACM **51**, 1, 107–113 (2008).

[12] Eldawy, A., "Spatialhadoop: towards flexible and scalable spatial processing using mapreduce", in "Proceedings of the 2014 SIGMOD PhD symposium", pp. 46–50 (ACM, 2014).

[13] Guttman, A., *R-trees: A dynamic index structure for spatial searching*, vol. 14 (ACM, 1984).

[14] Hartigan, J. A. and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm", Journal of the Royal Statistical Society. Series C (Applied Statistics) **28**, 1, 100–108 (1979).

[15] Iyer, A. P. and I. Stoica, "A scalable distributed spatial index for the internet-of-things", in "Proceedings of the 2017 Symposium on Cloud Computing", pp. 548–560 (ACM, 2017).

[16] Kim, J., S.-G. Kim and B. Nam, "Parallel multi-dimensional range query processing with r-trees on gpu", Journal of Parallel and Distributed Computing **73**, 8, 1195–1207 (2013).

[17] Kobren, A., N. Monath, A. Krishnamurthy and A. McCallum, "An online hierarchical algorithm for extreme clustering", arXiv preprint arXiv:1704.01858 (2017).

[18] Kothuri, R. K. V., S. Ravada and D. Abugov, "Quadtree and r-tree indexes in oracle spatial: a comparison using gis data", in "Proceedings of the 2002 ACM SIGMOD international conference on Management of data", pp. 546–557 (ACM, 2002).

[19] Liu, Y., W. Wei, A. Sun and C. Miao, "Exploiting geographical neighborhood characteristics for location recommendation", in "Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management", pp. 739–748 (ACM, 2014).

[20] Luo, L., M. D. Wong and L. Leong, "Parallel implementation of r-trees on the gpu", in "Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific", pp. 353–358 (IEEE, 2012).

[21] Murtagh, F., "A survey of recent advances in hierarchical clustering algorithms", The Computer Journal **26**, 4, 354–359 (1983).

[22] Nagarkar, P. and K. S. Candan, "Hcs: Hierarchical cut selection for efficiently processing queries on data columns using hierarchical bitmap indices.", in "EDBT", pp. 271–282 (2014).

[23] Nagarkar, P., K. S. Candan and A. Bhat, "Compressed spatial hierarchical bitmap (cshb) indexes for efficiently processing spatial range query workloads", Proceedings of the VLDB Endowment **8**, 12, 1382–1393 (2015).

[24] Samet, H., "Region representation: quadtrees from binary arrays", Computer Graphics and Image Processing **13**, 1, 88–93 (1980).

[25] Samet, H., *Foundations of multidimensional and metric data structures* (Morgan Kaufmann, 2006).

[26] Steinbach, M., G. Karypis, V. Kumar *et al.*, "A comparison of document clustering techniques", in "KDD workshop on text mining", vol. 400, pp. 525–526 (Boston, 2000).

[27] Wang, F., A. Aji and H. Vo, "High performance spatial queries for spatial big data: from medical imaging to gis", Sigspatial Special **6**, 3, 11–18 (2015).

[28] You, S., J. Zhang and L. Gruenwald, “Large-scale spatial join query processing in cloud”, in “2015 31st IEEE International Conference on Data Engineering Workshops (ICDEW)”, pp. 34–41 (IEEE, 2015).

[29] Yu, J., J. Wu and M. Sarwat, “Geospark: A cluster computing framework for processing large-scale spatial data”, in “Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems”, p. 70 (ACM, 2015).

[30] Zaharia, M., M. Chowdhury, M. J. Franklin, S. Shenker and I. Stoica, “Spark: Cluster computing with working sets.”, HotCloud **10**, 10-10, 95 (2010).

[31] Zhong, Y., J. Han, T. Zhang, Z. Li, J. Fang and G. Chen, “Towards parallel spatial query processing for big spatial data”, in “Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International”, pp. 2085–2094 (IEEE, 2012).

APPENDIX A

LIST OF NOTATIONS AND ABBREVIATIONS

N	number of queries	8
P	Partition set	9
Q	Query workload	8
Q_{size}	Query workload size i.e. set of queries in the workload	8
q	range query	8
M	number of partitions in partition set	9
p_i	single partition in partition set	9
C	Initial cost of processing queries in the node. This does not include the discount and tax charges.	26
c	cost of processing a single query	26
C_p	cost of processing a node with discount and tax	27
E_p	Empty area percent	32
O_p	Overlap area percent	32
B_{exp}	Expected budget in each partition	34
$C_P.root$	Projected cost at root node	34