Indexing Multi-dimensional Stream Data in a Cloud System

Afsin Akdogan, Ugur Demiryurek, Cyrus Shahabi

Integrated Media Systems Center

University of Southern California

ICampus ✓ IWatch CT ✓

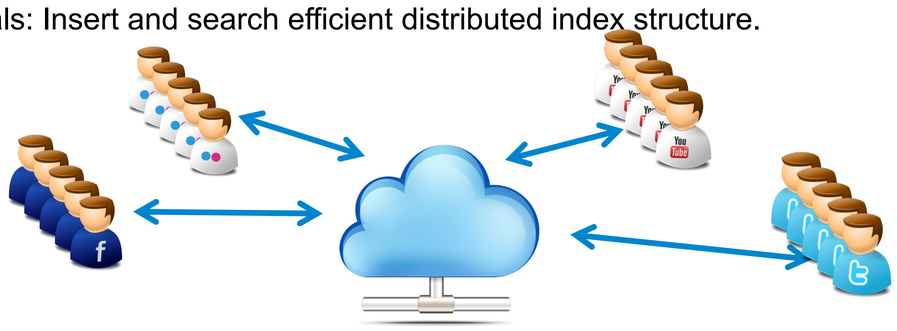
Introduction

• The amount of geospatial data is rapidly growing and geospatial queries are time consuming problems especially with large datasets.

Application

• **Internet-scale** applications, where hundreds of servers are used to support terabytes of data and millions of users.

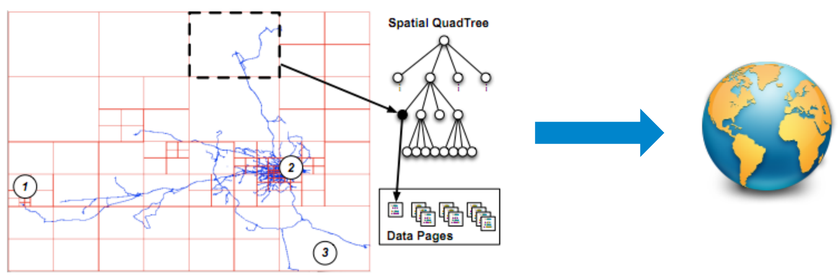
• Goals: Insert and search efficient distributed index structure.

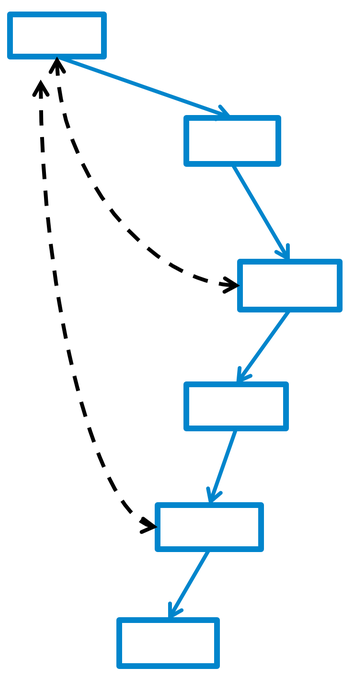


Chord-Quadtree

• Indexing space by Quadtree to handle fast insertion rate.

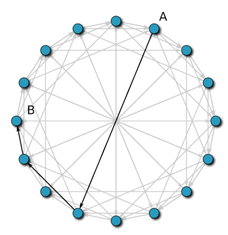
• Quadtree might have long paths due to **non-uniform** data.





• Store a routing table at each node for faster search.

• Only split causes update of the routing tables which is very infrequent.



Advantages

• It can efficiently handle frequent inserts.

• It finds successor node in **O(log *N*)** where there are N nodes in a path from root to leaf. Fast search.

• No top-down search. We convert the Quadtree into an undirected graph. Better **load-balancing** due to the fact that any node can start the query.

Related Work

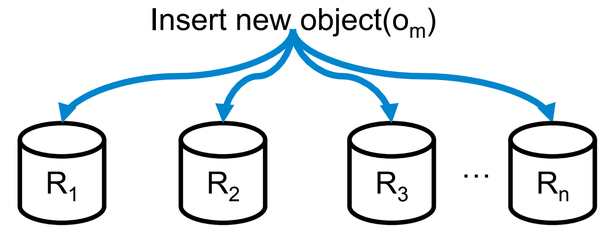
Centralized Algorithms

• Full replication of data.

⛔ Cannot handle **dynamic** data. Update/insert should be done in all machines.

⛔ Doesn't work if the data don't fit in a single machine.

⛔ Doesn't take advantage of GBytes of distributed RAM.



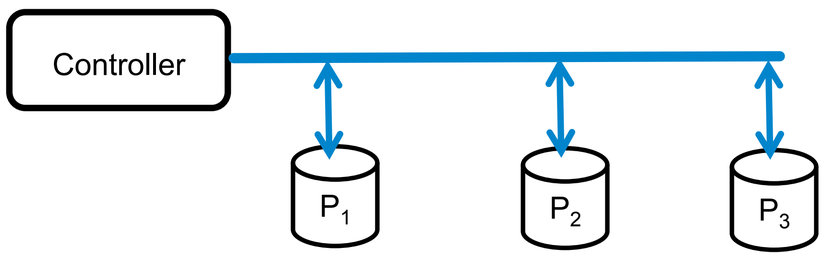
Parallel Databases

⛔ Not completely decentralized. **Not scalable**. Overloads controller.

• Controller keeps a global index. Global index is proportional to the data size of the data. It doesn't scale well.

• Inserts/Updates on the data should be sent to the controller too.

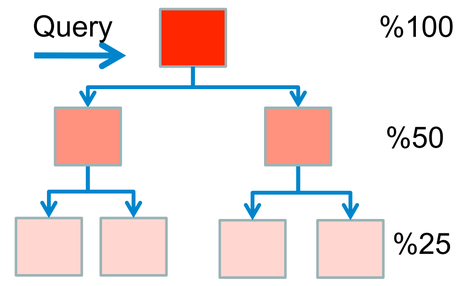
• All requests go to controller first since it manages query answering process.



Distributed Index Structures

⛔ Top-down search overloads the nodes near the tree root.

⛔ The balancing needs to fully rebuild the tree using multicast from all servers (distributed kd-tree).



✅ Traverse the tree from bottom-up. Avoids hotspots.

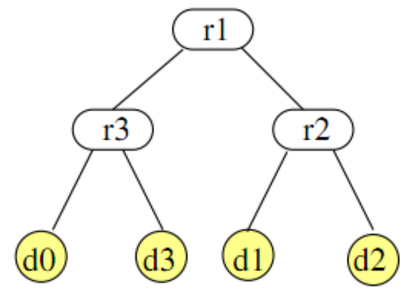
⛔ Too costly to maintain the index for frequent inserts.

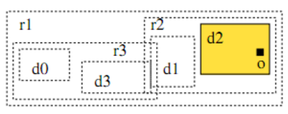
Insertion cost

Overlapping Coverage Cost

Split Cost

Balancing cost





Future Work

• Support point and window queries as the other approaches explained above since those are the queries used by internet applications the most.

• Conduct extensive experiments to test:

• Insert/Update performance of the index.

• Search performance of the index in the presence of millions of concurrent queries.