Distributed Spatial Indexing for Scientific Datasets

Beomseok Nam

CMSC 818s

Peer-to-Peer and Grid Computing

May. 10, 2007

Motivation of Distributed Spatial Indexing

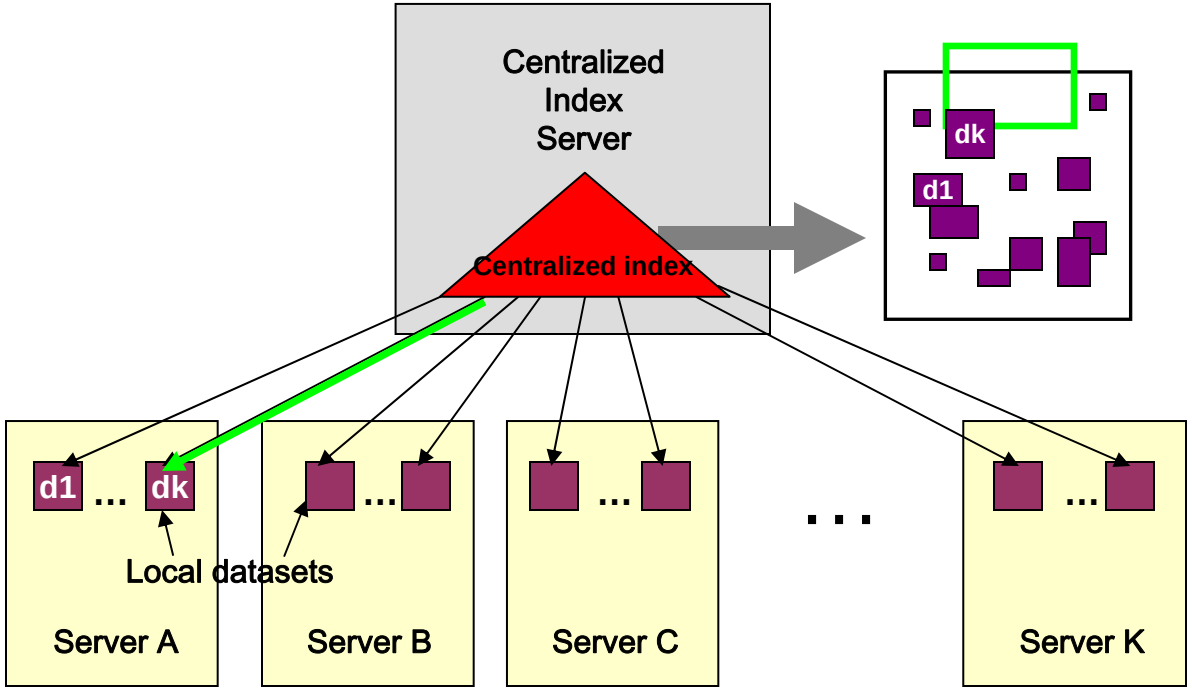
- Properties of Scientific Applications
  - Generate multidimensional datasets (i.e, space and time)
  - Generate very large datasets at fast rate
  - Data/Computation intensive
  - Datasets are distributed

- Access pattern into Scientific Datasets
  - Multidimensional Range Query
    - Retrieves data that overlaps given range of values
    - Ex) SELECT temperature
      FROM dataset
      WHERE (latitude BETWEEN 20 AND 30)
      AND (longitude BETWEEN 50 AND 60)

Indexing Distributed Datasets

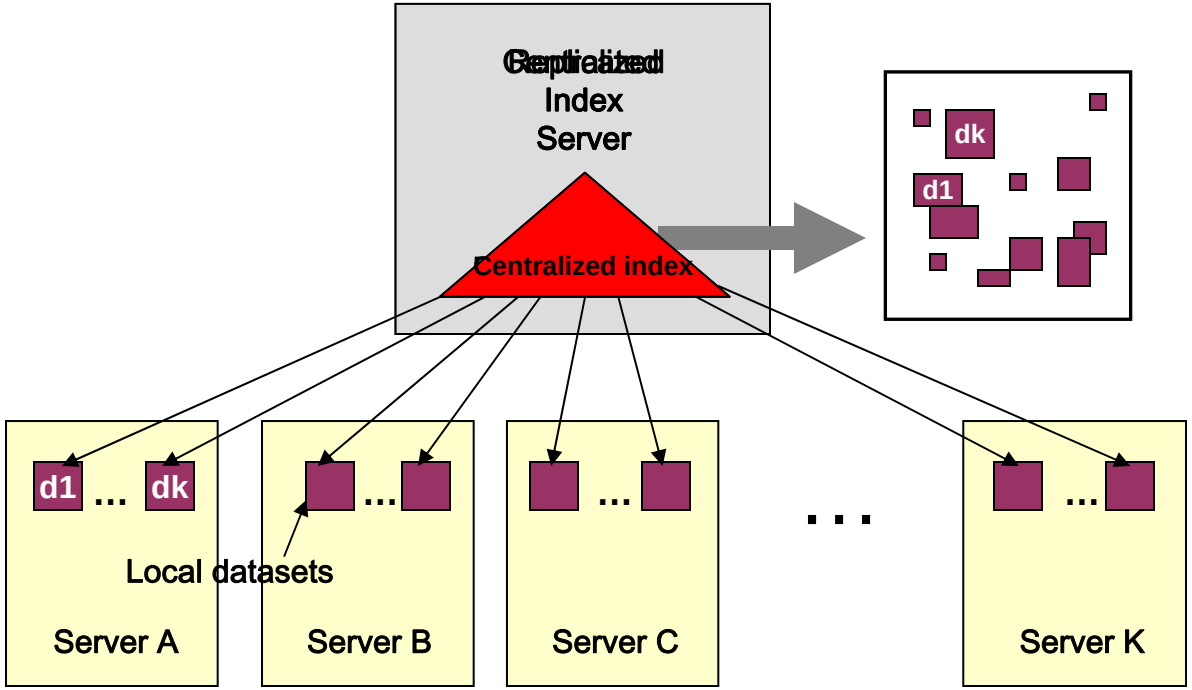
- Multidimensional indexing trees
  - k-d-Trees, R-Trees, etc
  - Q: Is a single index enough for distributed large scientific datasets?
- Distributed indexing is better than a centralized single index
  - Performance & Scalability
- Q: How to distribute multidimensional index?
  - Replication
  - Hierarchical Two Level Indexing
  - Decentralized Two Level Indexing

Centralized Indexing



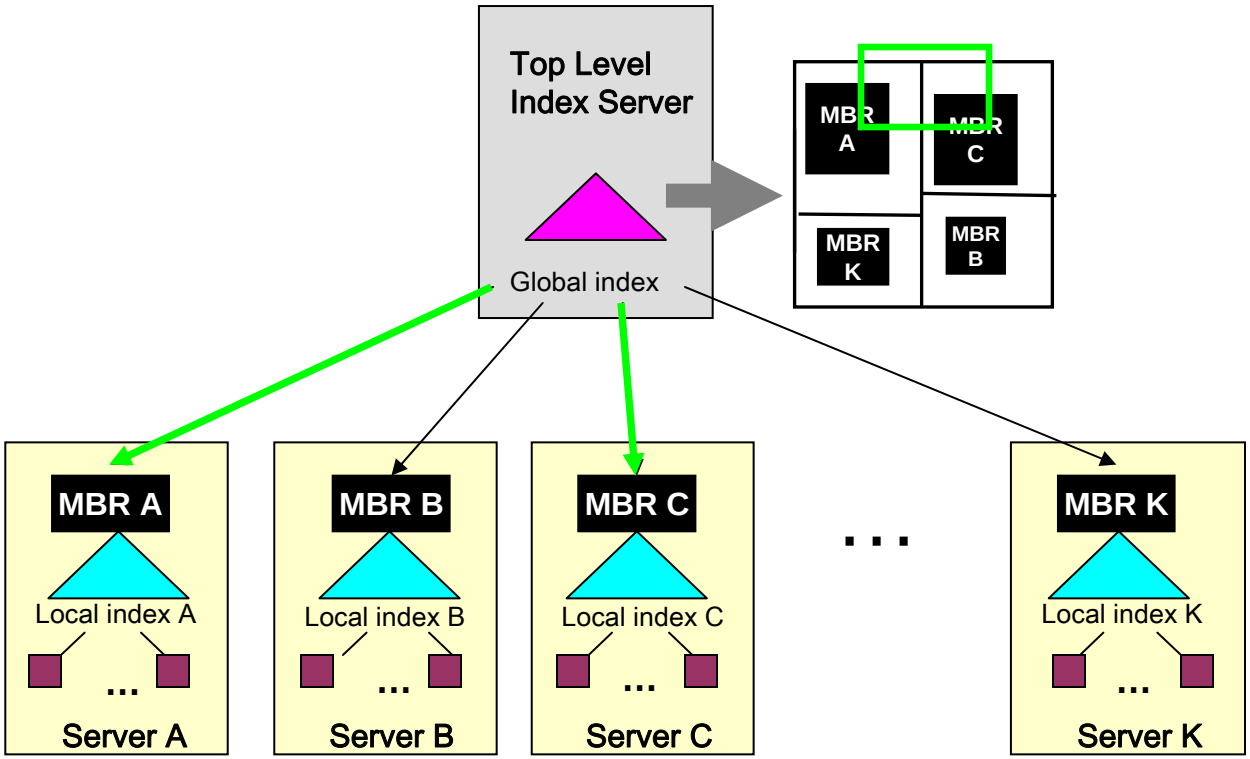
- Commonly used
- Easy to implement
- Many drawbacks
  - Network latency, resource bottleneck, single point of failure

Replicated Indexing



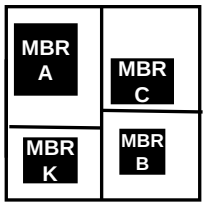
- Replicate a centralized index that stores all the distributed data information
- Avoid resource bottleneck & single point of failure
- Expensive index update
  - Consistency Issues

Hierarchical Two Level Indexing

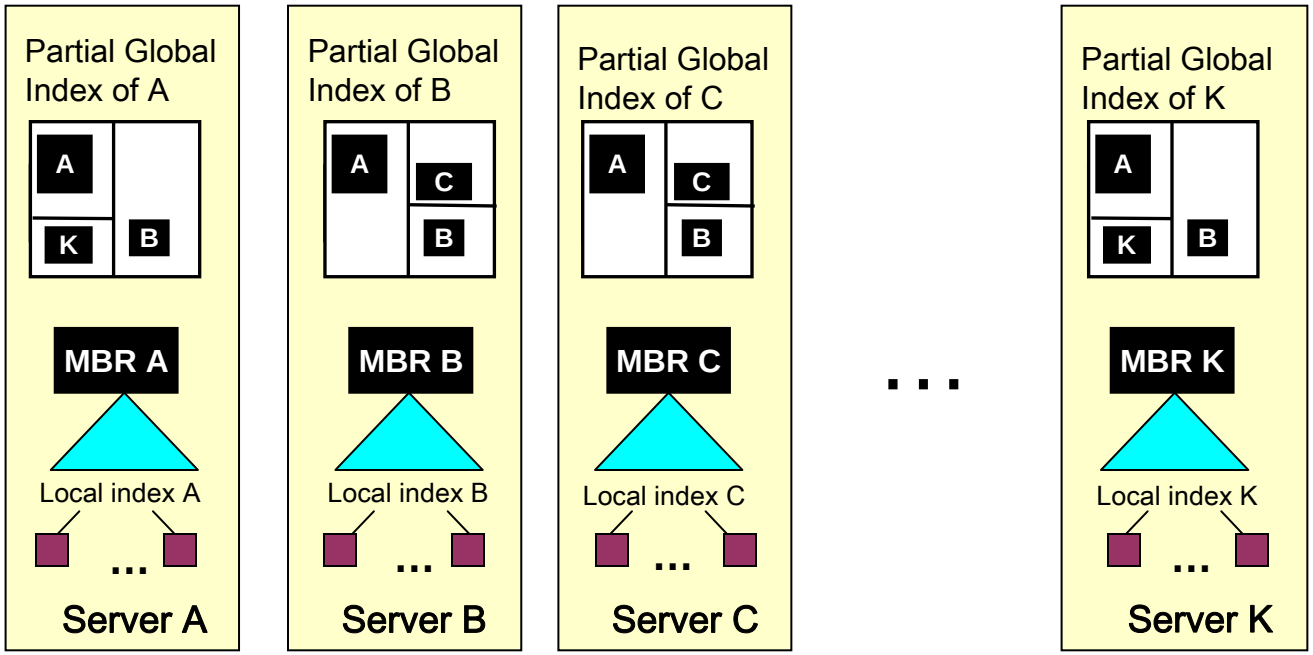


- Local index for local datasets
- Global index to determine which local index must be accessed
- Partitioning index to parallelize index operations
  - Smaller index → faster performance
  - Insertion is done locally in most cases

Decentralized Two Level Indexing



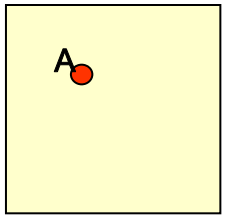
: complete global index is stored nowhere
: No global index server, but partial global index in each data server



- Eliminates any potential central bottleneck in p2p fashion
- Routing & join algorithms are needed
- *DiST* (Distributed Search Tree)

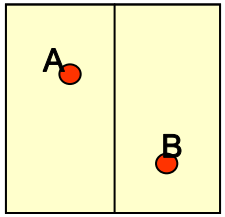
DiST: Server Join

Event 1)
Server A joins



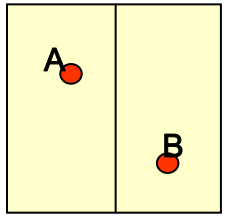
Server A's
partial global index (PGI)

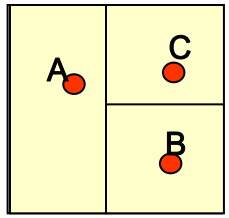
Event 2)
Server B joins
via server A



A's PGI **B's PGI**

Event 3)
Server C joins
via server B

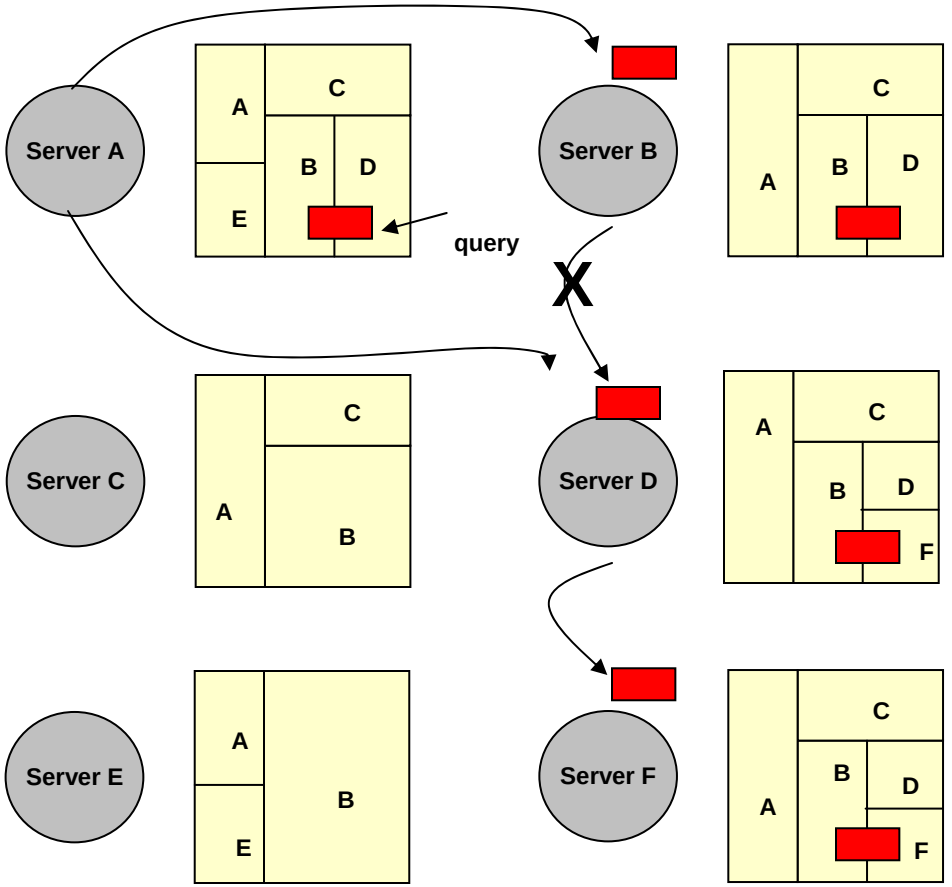




A's PGI **B's PGI** **C's PGI**

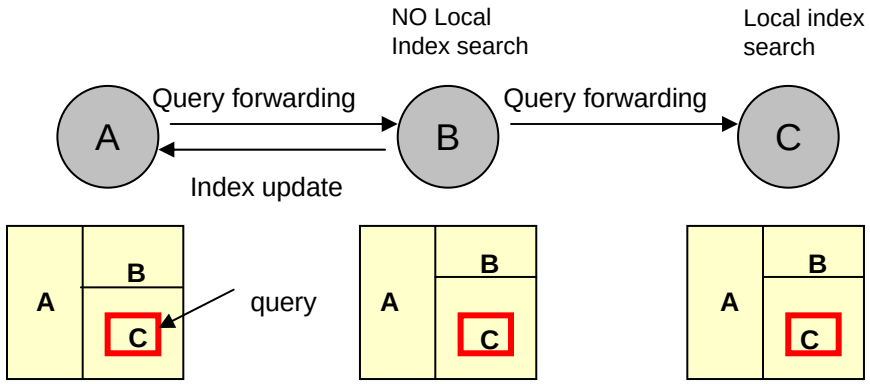
- Convert MBR into high dimensional point
- Each server has a partial global index represented as a KD-tree
- When a server receives a join
  - if the server's region contains newly joined server's MBR
    → divide its partition as for KD-trees
  - Otherwise, forward it to another server

DiST: Query Routing



- Query routing with partial global index will eventually forward queries to actual destination servers
- Query forwarding history is required
  - to check whether query originating server received all the results
  - to eliminate duplicate query processing
- Unbalanced global index may cause a long message chain

DiST: Lazy Index Update

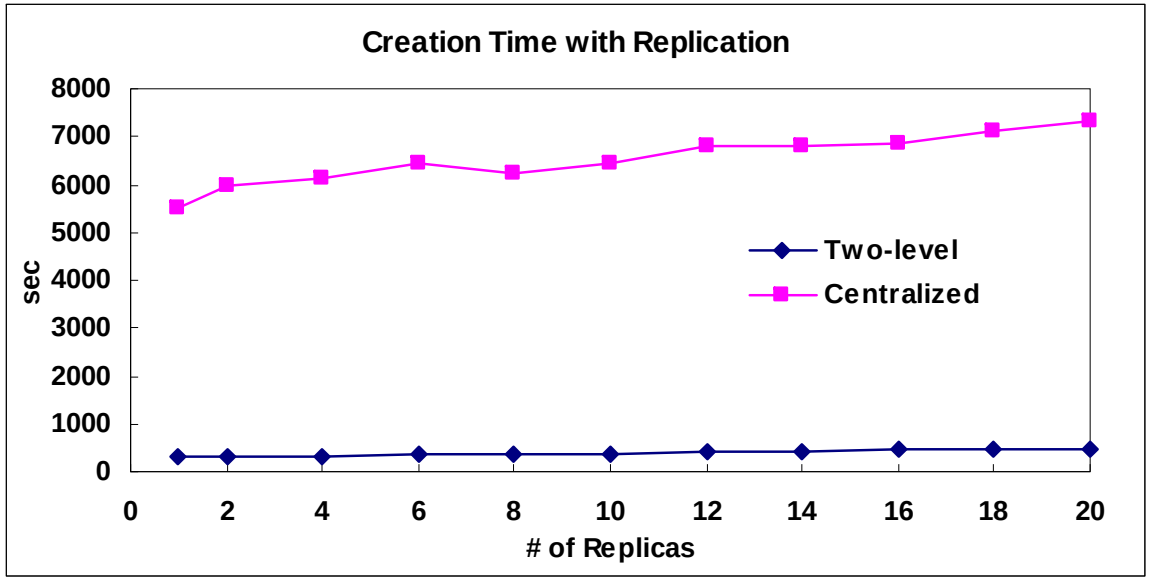


- Any partial global indexes can be merged to each other
  - There exists only one complete global index
  - Partial global indexes are missing some leaf nodes
- Lazy index update
  - triggered when a server received a query but it has to forward the query to another server not in query forwarding history
  - Eventually all partial global indexes converge to the same state

Experiments (Replica vs Hierarchical)

- Platform: Storage Resource Broker (SRB)
  - Storage middleware in GRID environment
  - Implemented multidimensional indexing service on top of SRB
  - MCAT (Metadata Catalog) bottleneck
    - Did not register index files in MCAT
- Experiment Environment
  - 40 Linux nodes (rogue nodes)
    - P III 650MHz
    - 100Mb/s Ethernet
- Datasets
  - AVHRR GAC level 1B satellite datasets
  - 3 dimensions (Latitude, Longitude, Time)
  - One month of data (30GB)
  - Partitioned into 400,000 chunks
  - Index MBR of each chunk
  - 10,000 chunks in each of 40 nodes

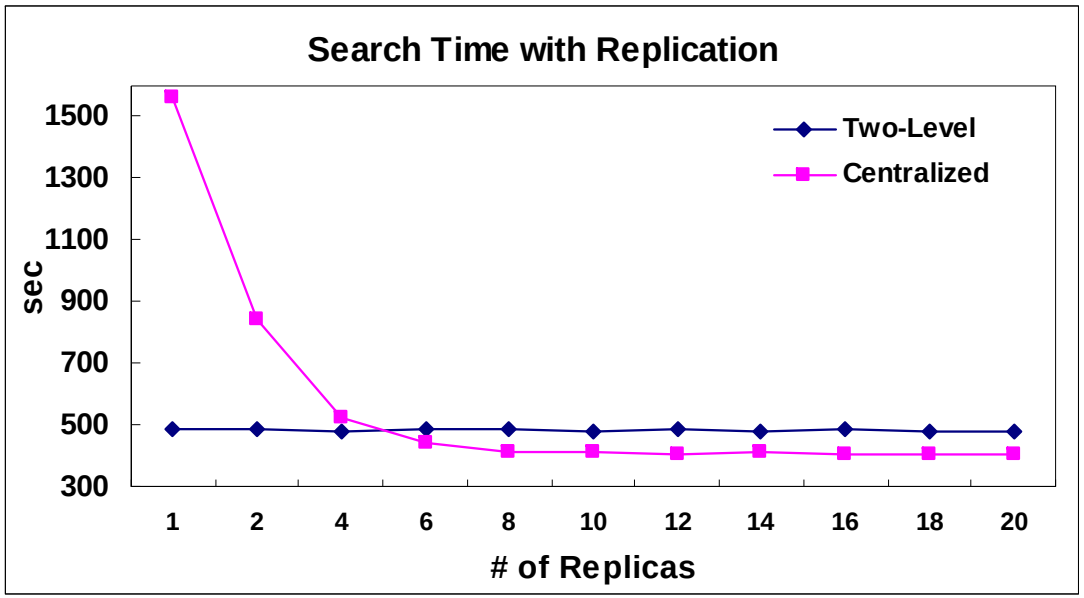
Index Creation with Replicas



- 40 nodes
- Each node has a single client that inserts 10,000 data objects

- Replication hurts the insertion performance a little bit
  - We do not enforce strong consistency
  - 32% increase in centralized index (20 replicas)
  - 58% increase in two-level index, from a much lower starting point

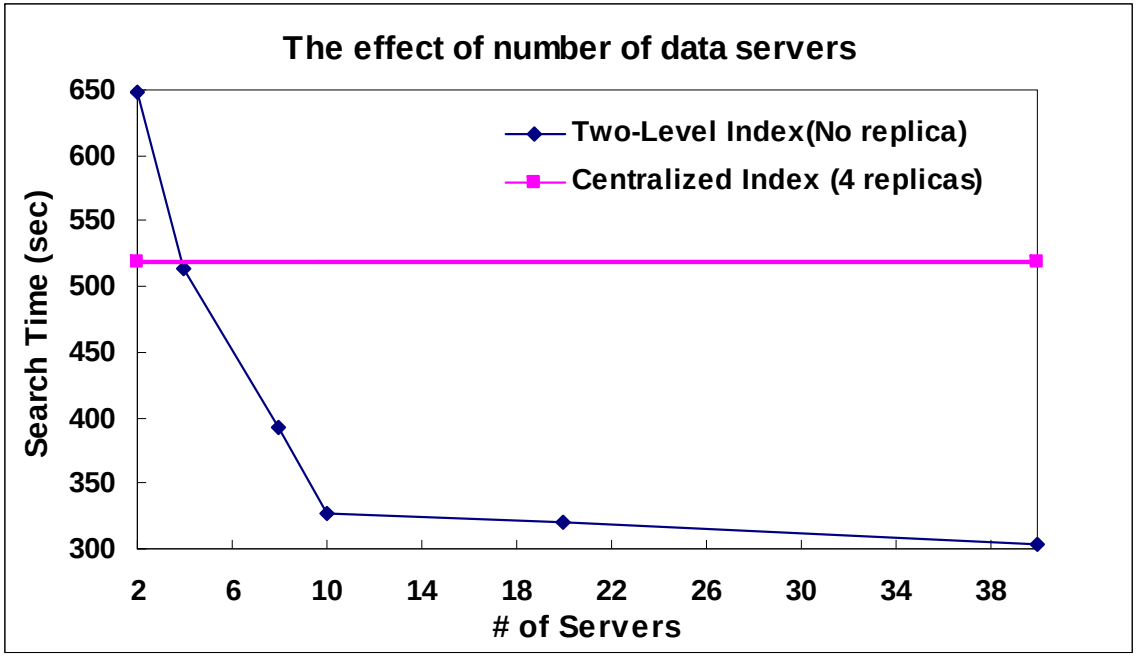
Index Search with Replicas



- 40 nodes
- Each node has a single client that submits 100 queries

- As there are more replicas
  - Replicated centralized index distributes the workload onto more replica servers
  - The load of global index server in two-level index is not so heavy compared to centralized indexing

Search performance varying # of servers

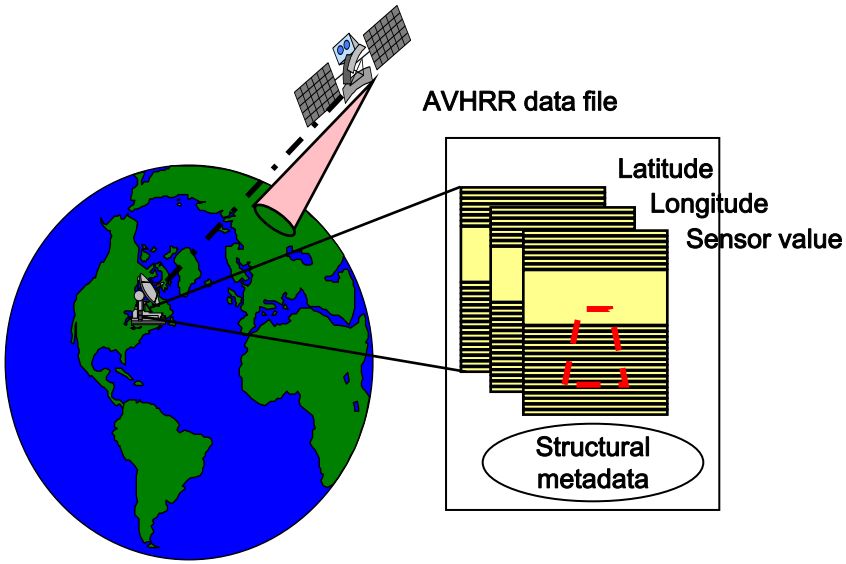


- 2-40 nodes
- 400,000 total data objects
- 200,000 on 2 nodes down to 10,000 on 40 nodes
- 40 clients submit 100 queries each

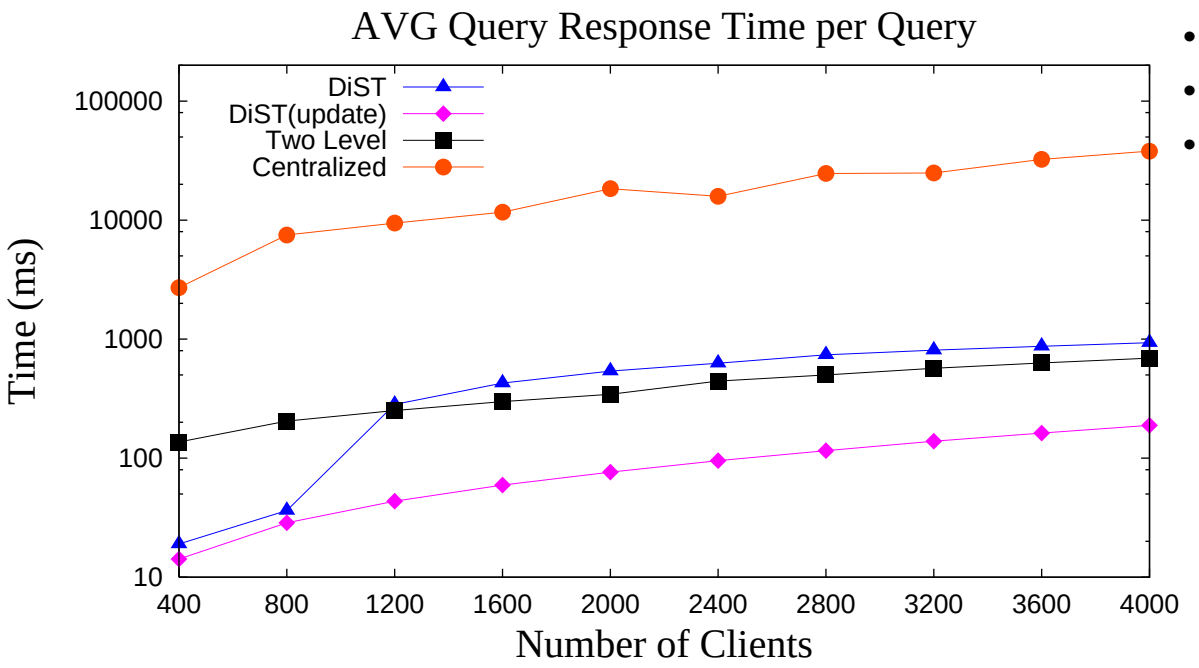
- As there are more data servers,
  - Two-level index distributes the load of index searching onto more servers
  - Centralized indexing does not distribute the workload

Experiments (Hierarchical vs DiST)

- Experimental Environment
  - 41 Linux cluster nodes (rogue nodes)
    - Intel Xeon 2.66GHz
  - Intercommunication
    - Myrinet (1Gb/s transfer rate)
    - TCP socket
- Datasets
  - AVHRR GAC level 1B satellite datasets
  - 3 dimensions (Latitude, Longitude, Time)
  - One month of data (30GB)
  - Partitioned into 120,000 chunks
  - Index MBR of each chunk
  - 3,000 chunks in each node
- Synthetic Query Workload
  - Customer Behavior Model Graph (CBMG)
  - New point of interest - 200 Hot spots
  - Spatio-temporal movement
  - Resolution increase/decrease



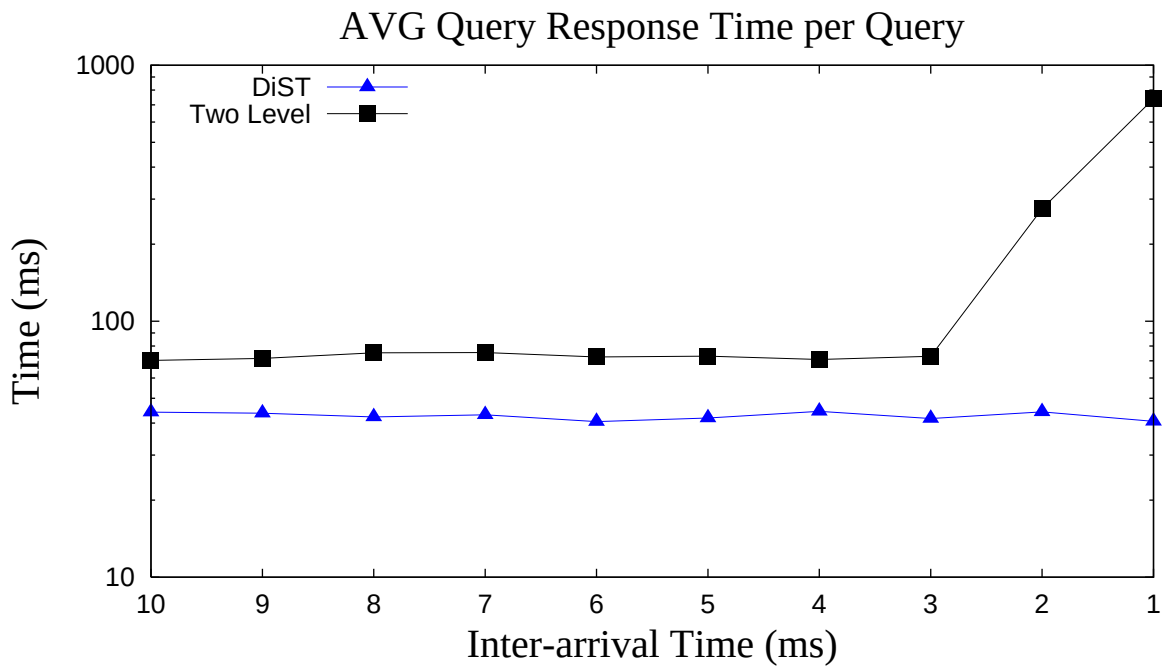
Scaling # of Clients



- 1000 sequential queries per client
- 41 servers
- QRT is the time from the moment a query is submitted until it completes

- Centralized indexing is the worst due to its huge size
- DiST without update is worse than hierarchical indexing when servers are heavily loaded
  - Longer routing path hurts more when system is busy

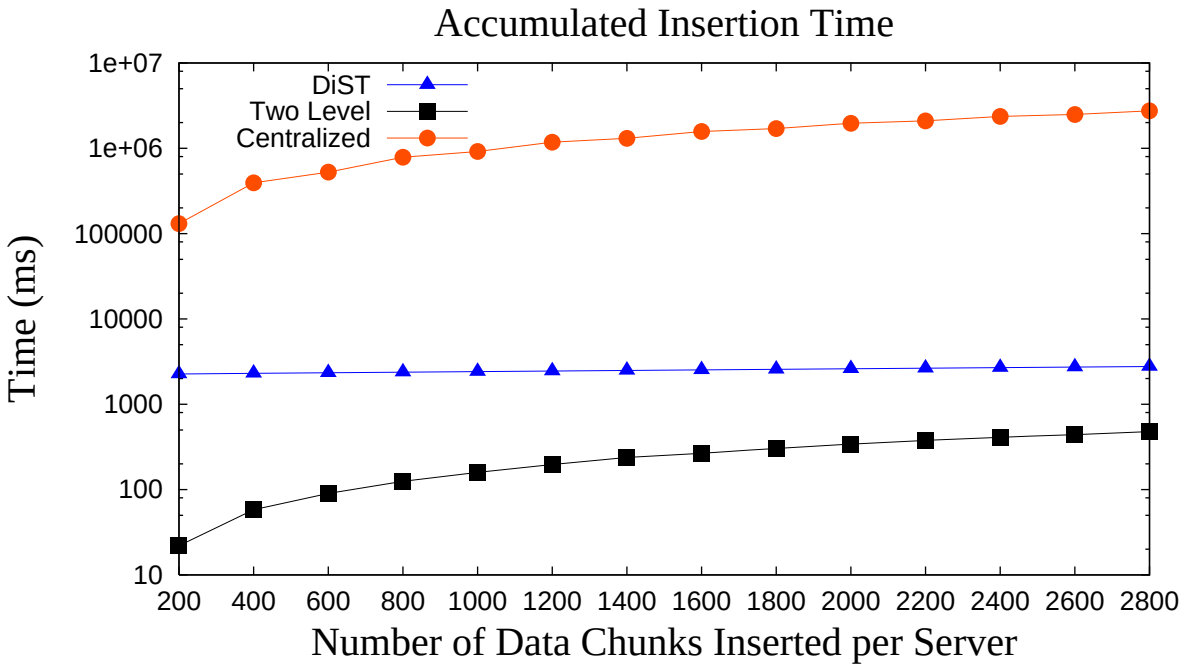
Simulated Search performance



- 1000 servers
- Avg packet delay between servers = 50ms
- Global Index lookup time is measured as a part of the simulation ≈ 2ms

- Large number of data servers leads to large global index
- When the system is heavily loaded, global index server becomes a performance bottleneck.

Insertion Time



- 41 servers
- Elapsed time to insert 3000 chunks

- Most insertions are done at local indexes for two level indexing and DiST
- Gap between DiST and hierarchical indexing comes from the overhead of DiST join algorithm

Summary

- Centralized single index
  - Performance bottleneck
  - Scalability problems
- Replication
  - Improves search performance
  - But does not alleviate insertion bottleneck
- Hierarchical two-level indexing
  - Works best for frequent index updates
- Decentralized two-level indexing
  - Scales well with a large number of servers

End of Talk