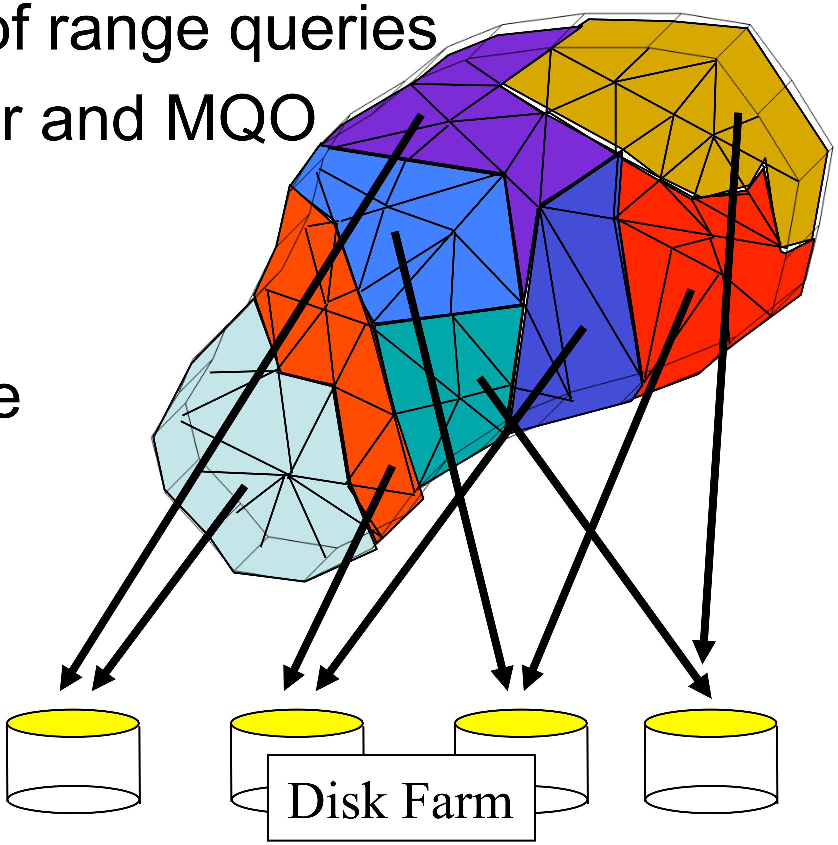
Effective Indexing of Distributed Multidimensional Scientific Datasets

Alan Sussman
Beomseok Nam
UNIVERSITY OF MARYLAND
Department of Computer Science &
Institutute for Advanced Computer Studies

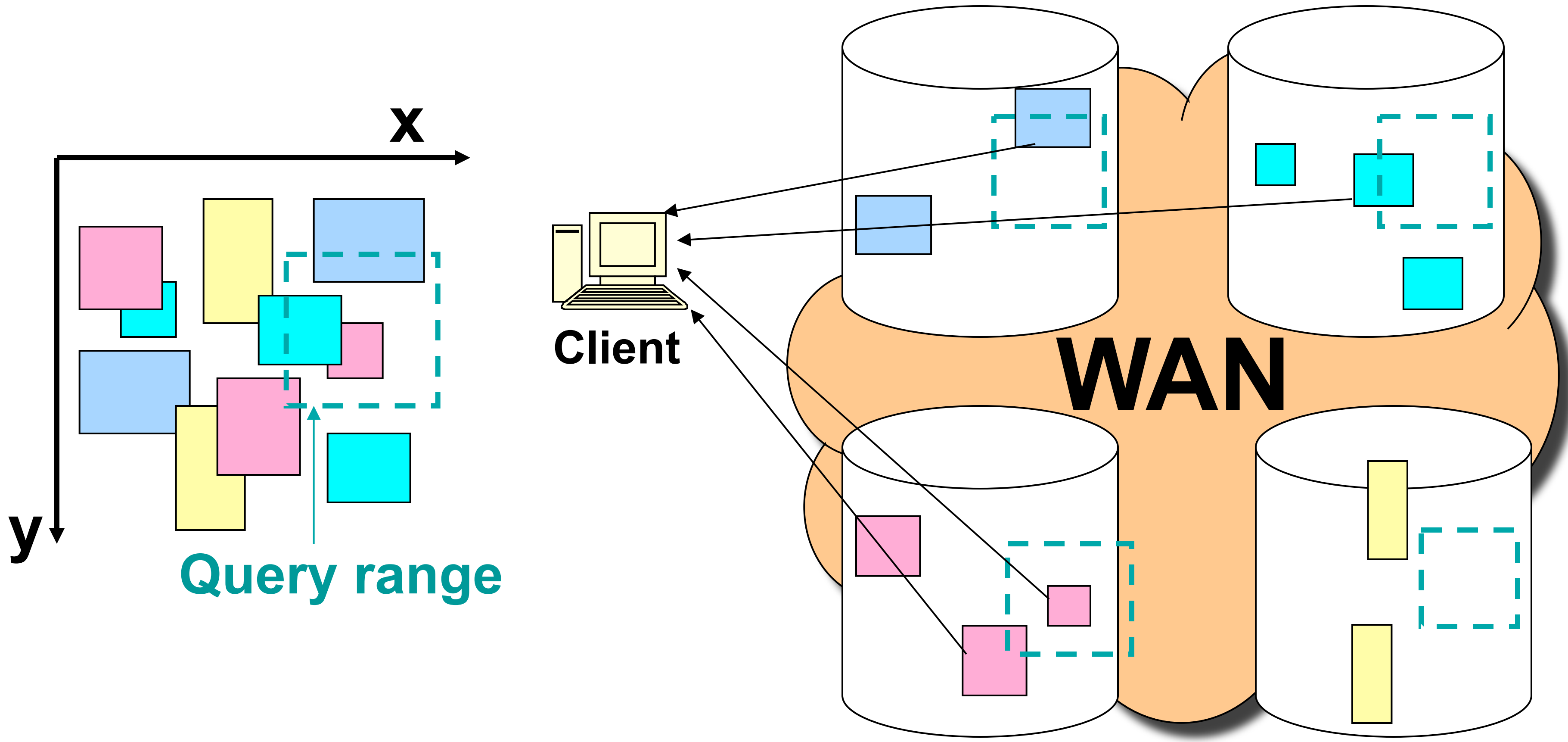
http://www.cs.umd.edu/projects/hpsl/chaos/ResearchAreas/gmil

Motivation

- How to deal with very large datasets?
  - Exploit parallelism across distributed systems, to accelerate the performance of range queries
  - Done in context of DataCutter and MQO systems at UMD
- Data Intensive Applications
  - Earth Science/Space Science
  - Medical Imaging
  - Oil Reservoir Modeling
  - Computer Vision



Range query across WAN



- Defines a region in a multi-dimensional space
- Extract data objects that have overlapping spatial coordinates

Overview

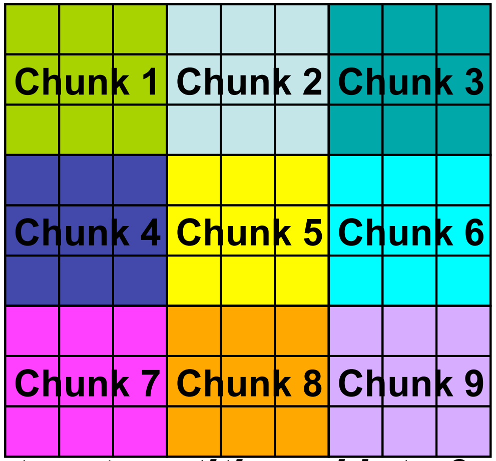
- How to index multidimensional datasets
  - self-describing data formats
  - chunking
  - data vs. space partitioning spatial indexing techniques
  - SH-trees
  - distributed datasets
- Distributing the index
  - Replication
  - Hierarchy
  - Full decentralization (not today)

Self-describing Scientific Data Formats

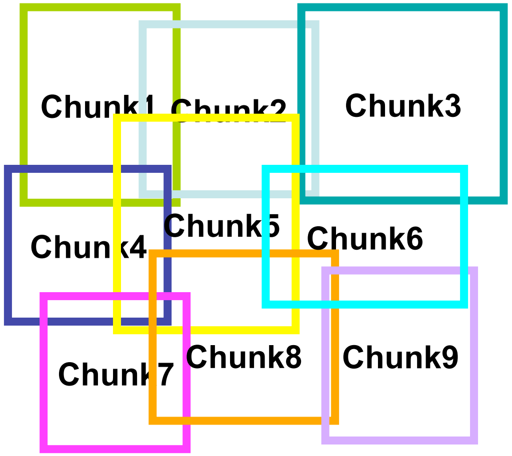
- netCDF, HDF4, HDF5, SILO, etc.
  - Common data formats in scientific computing
  - No outside information is needed to interpret them (syntactically)
  - Machine independent
  - No spatial indexing structures supported
- How to improve access to subsets of data?
  - Data chunking
  - Spatial indexing techniques

How to index scientific datasets?

- Scientific datasets
  - Collection of multidimensional arrays
  - Have spatial/temporal locality
    - Sensor devices store data in the order it is acquired, or simulations generate it that way
- Data Chunking
  - Partition a multidimensional dataset into coarse-grained hyper-rectangular blocks
    - Results in tight bounding boxes (MBRs) in problem space

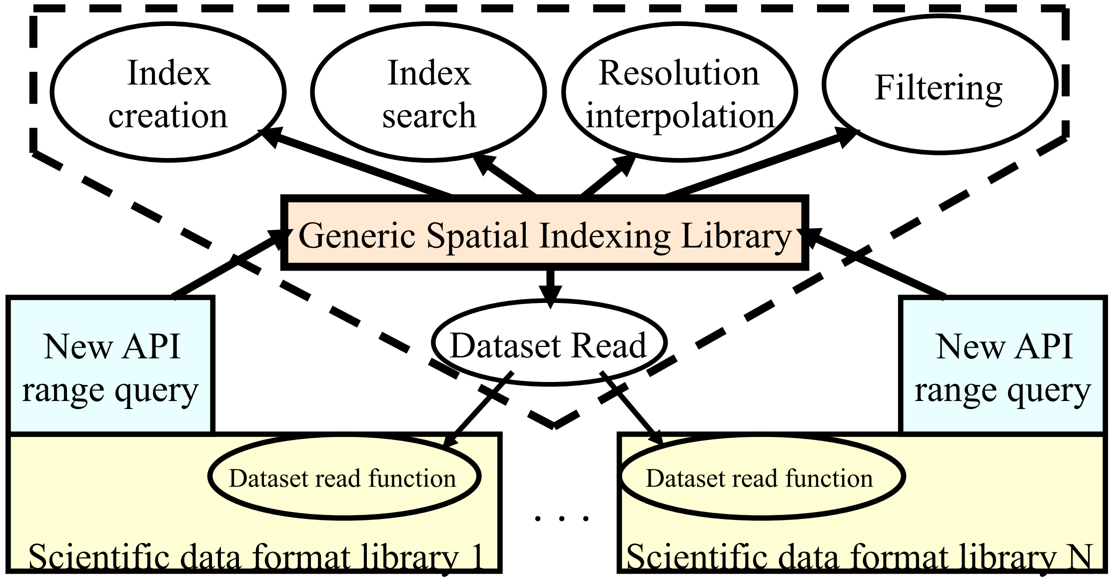


Dataset partitioned into 9 chunks (N dimensional array)



Problem space (M dimensional objects)

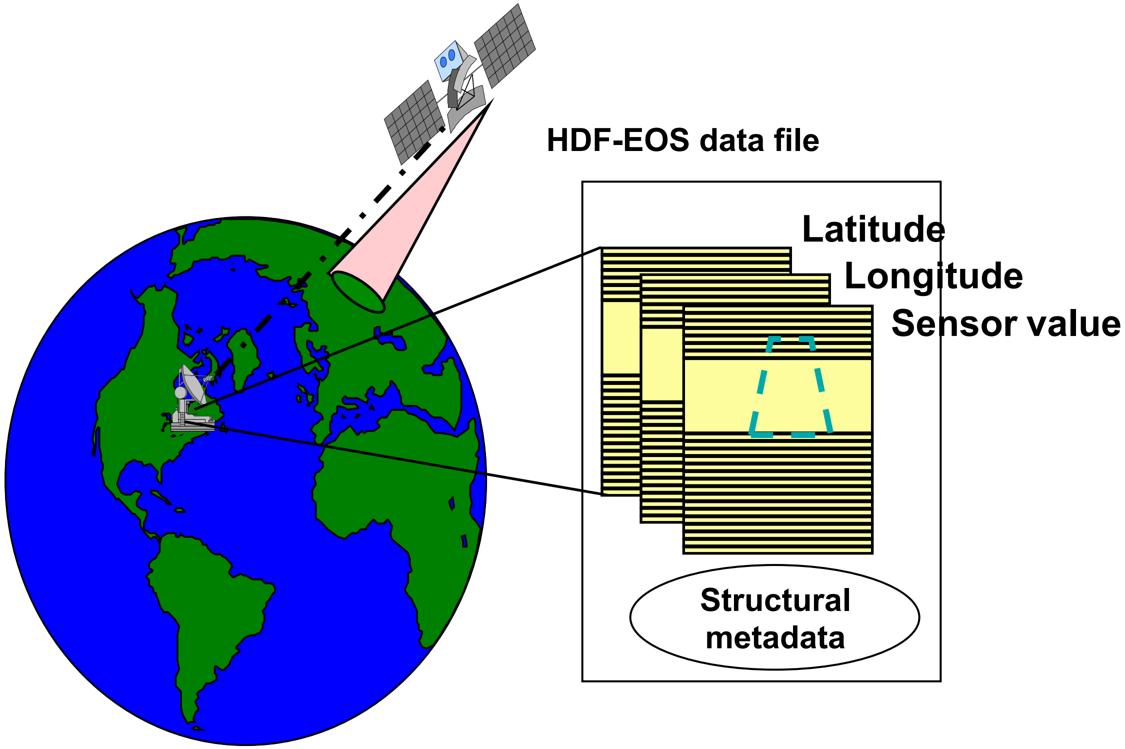
Generic Multidimensional Indexing Library (CCGrid2003, SC2003)



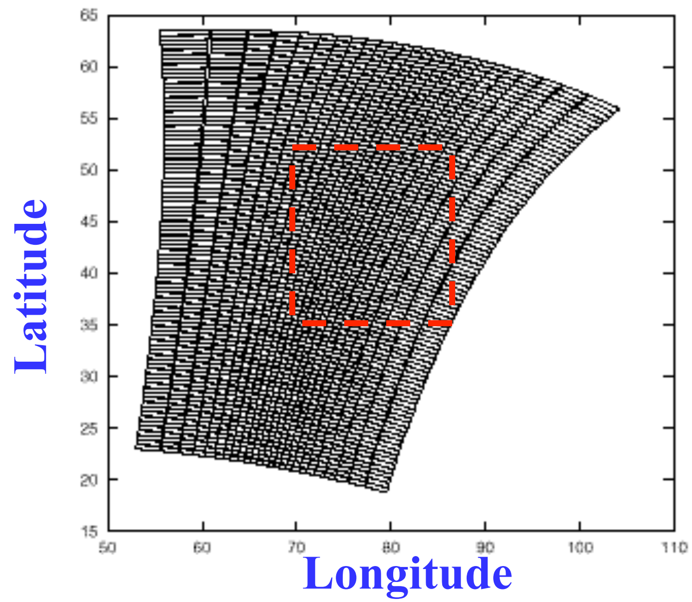
- Index accelerates range query performance
- Data chunking makes the index smaller and faster
  - Filtering out unneeded data has negligible overhead
- Scientific data format libraries (netCDF, HDF4, HDF5) do not have built-in indexing functions.
- GMIL provides indexing functions based on data chunking, on top of multiple scientific data format libraries

Case study: HDF-EOS

- NASA's Earth Observing System project
- Extended the capabilities of HDF, called *HDF-EOS*.



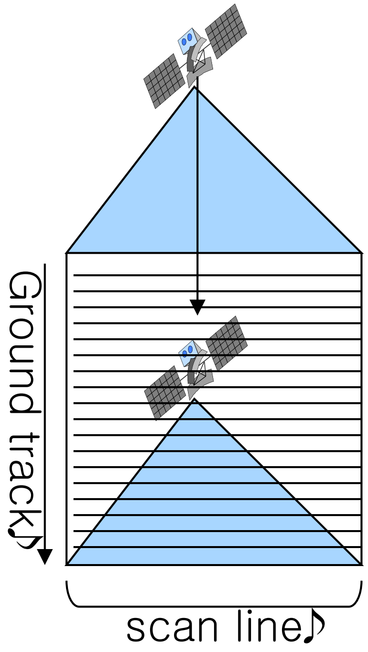
Spatial irregularity



- *HDF-EOS defboxregion* function must read every geographic scan line

Case study: HDF-EOS(cont.)

- Does not support spatial indexing structures.
  - *Any-point* : Compare every element with query range
  - Two approximation options to reduce high disk I/O

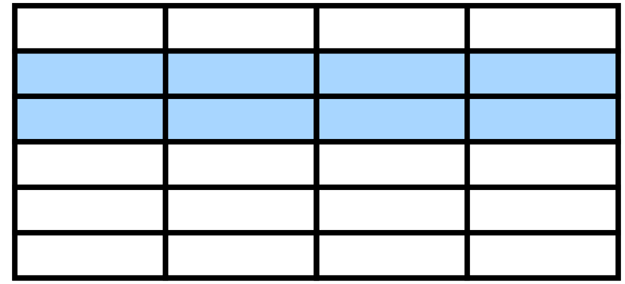


- HDF-EOS provides approximation options
  - *Mid-point* : Compare only midpoint of a scan line
  - *End-point* : Compare both ends of a scan line
- New HDF-EOS range query function
  - Utilizes generic spatial indexing library

Experimental Setup

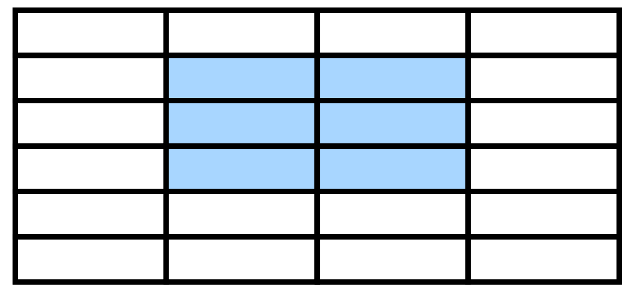
- To measure index creation time and range query time.
- Machine:
  - Sun Blade 100 workstation (a 500MHz Sparcv9 processor)
  - 256 MB memory
- Data:
  - Test dataset ranges from 16MB to 128MB
- Range Query:
  - Three different shapes
- H5Xread from GMIL instead of H5Dread
  - Uses R^* tree for index

Query 1



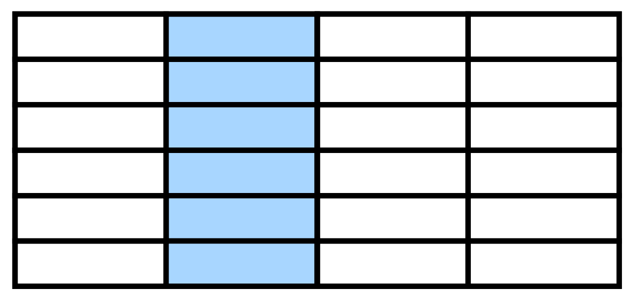
Selected region is long in row major order

Query 2



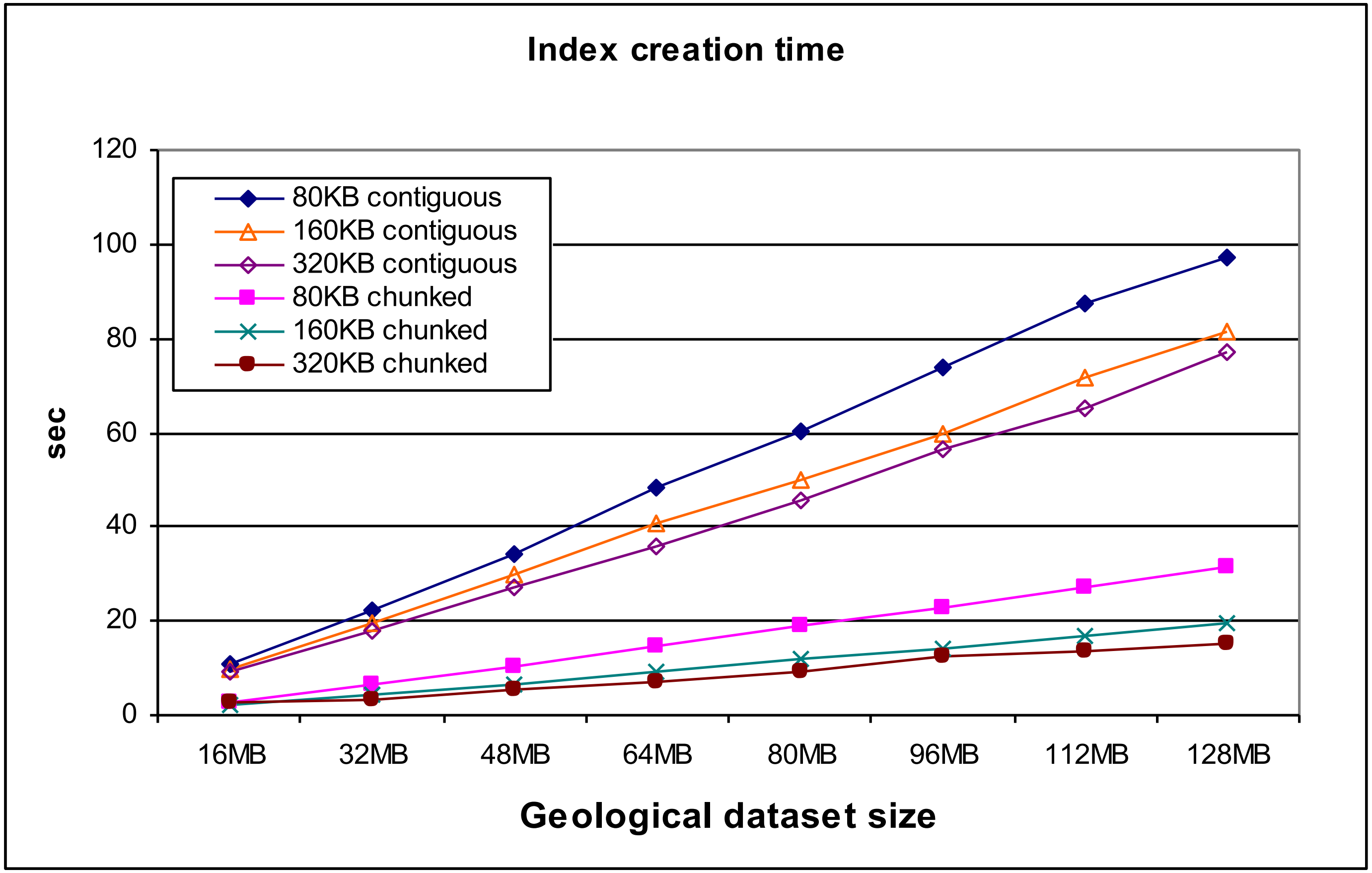
Selected region is almost square

Query 3



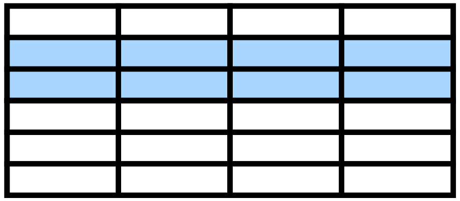
Selected region is long in column major order

Index creation time



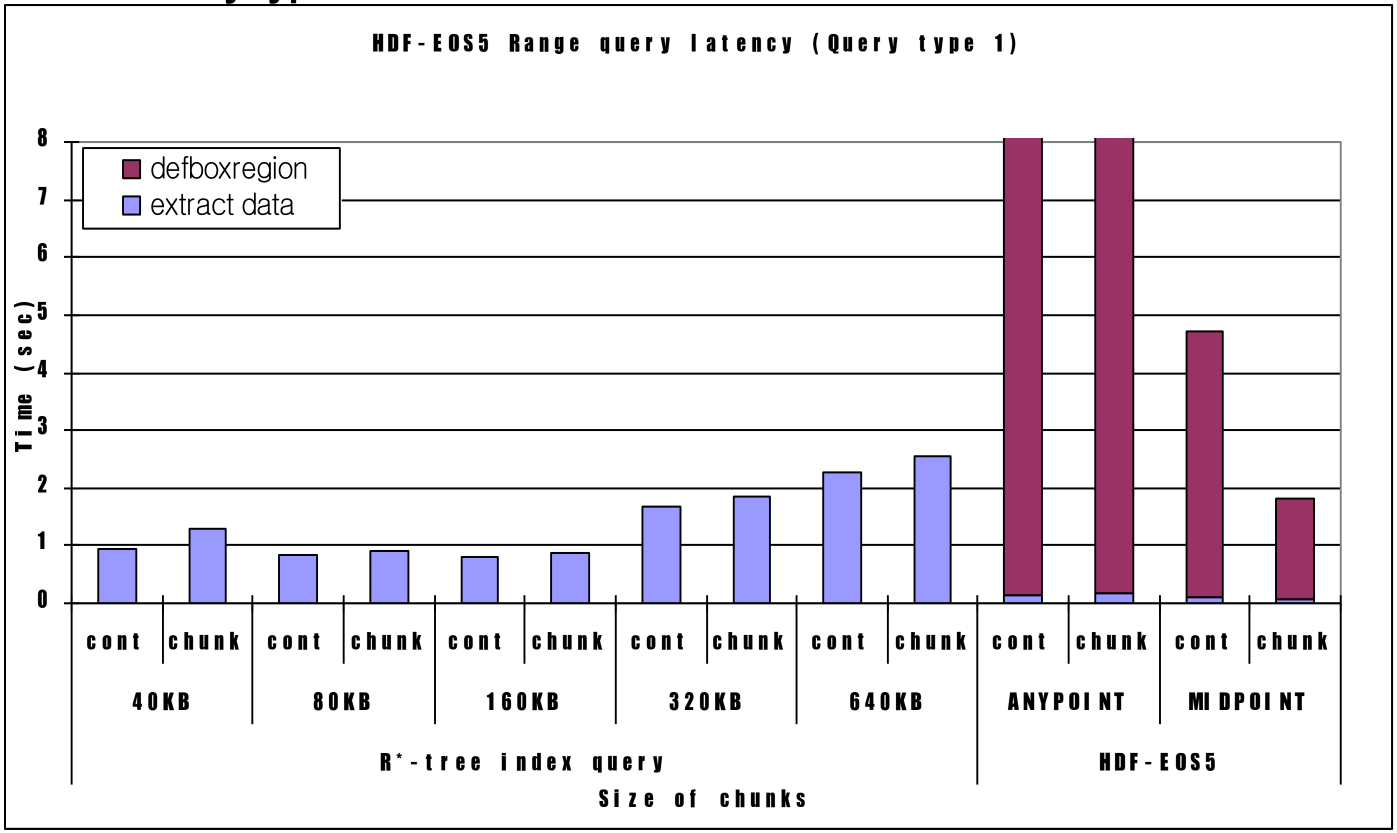
- Chunked layout shows better performance
- As the number of chunks grows, the time to create index increases

Range query performance (1)

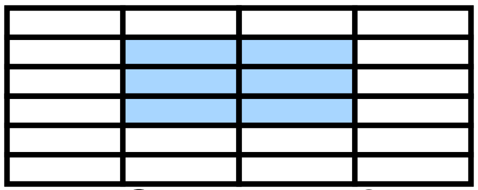


Query type 1

Time to read a region with many columns and relatively few rows

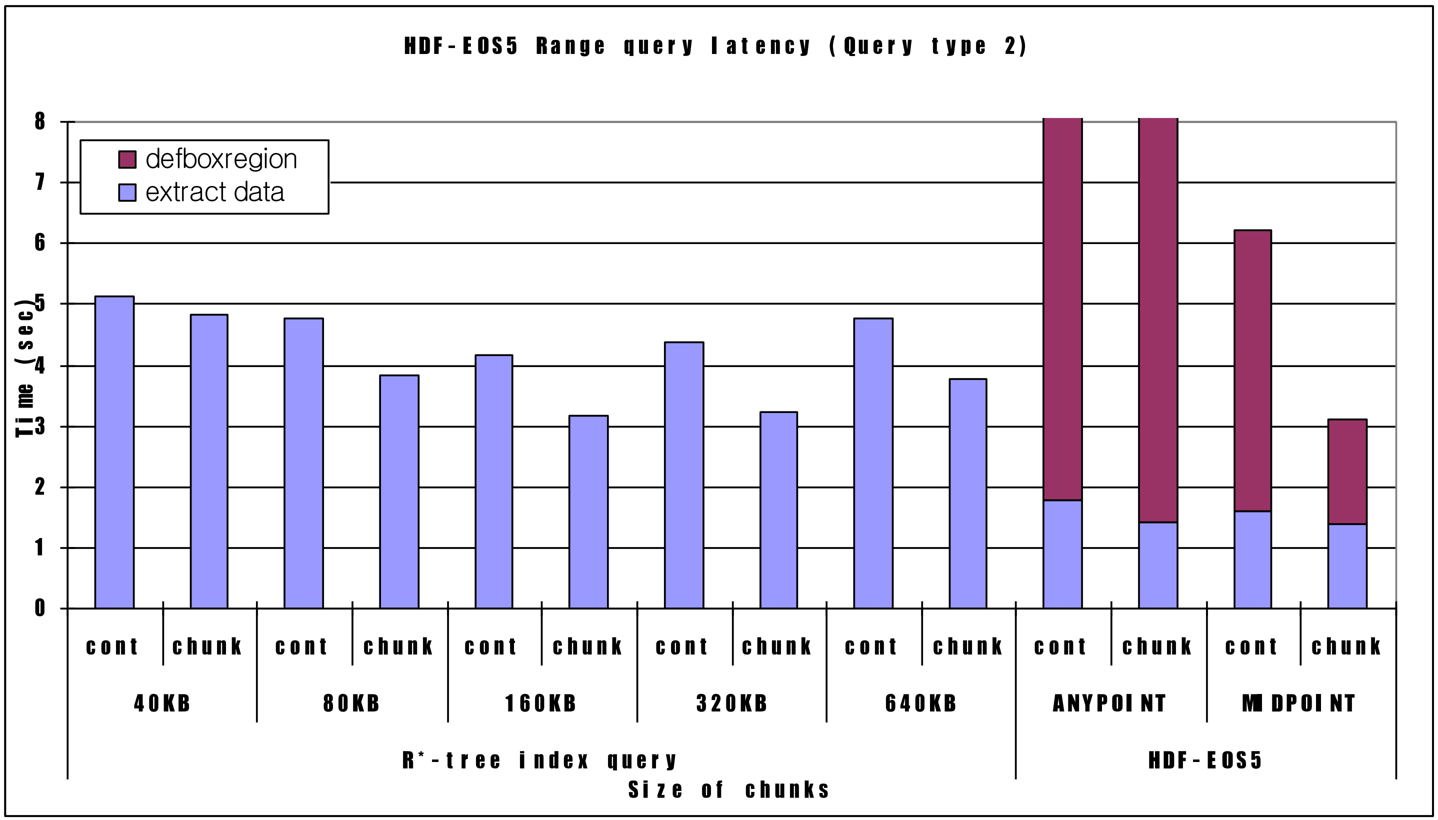


Range query performance (2)

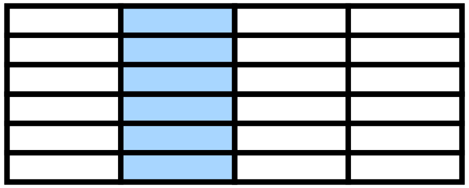


Query type 2

Time to read a mostly square region

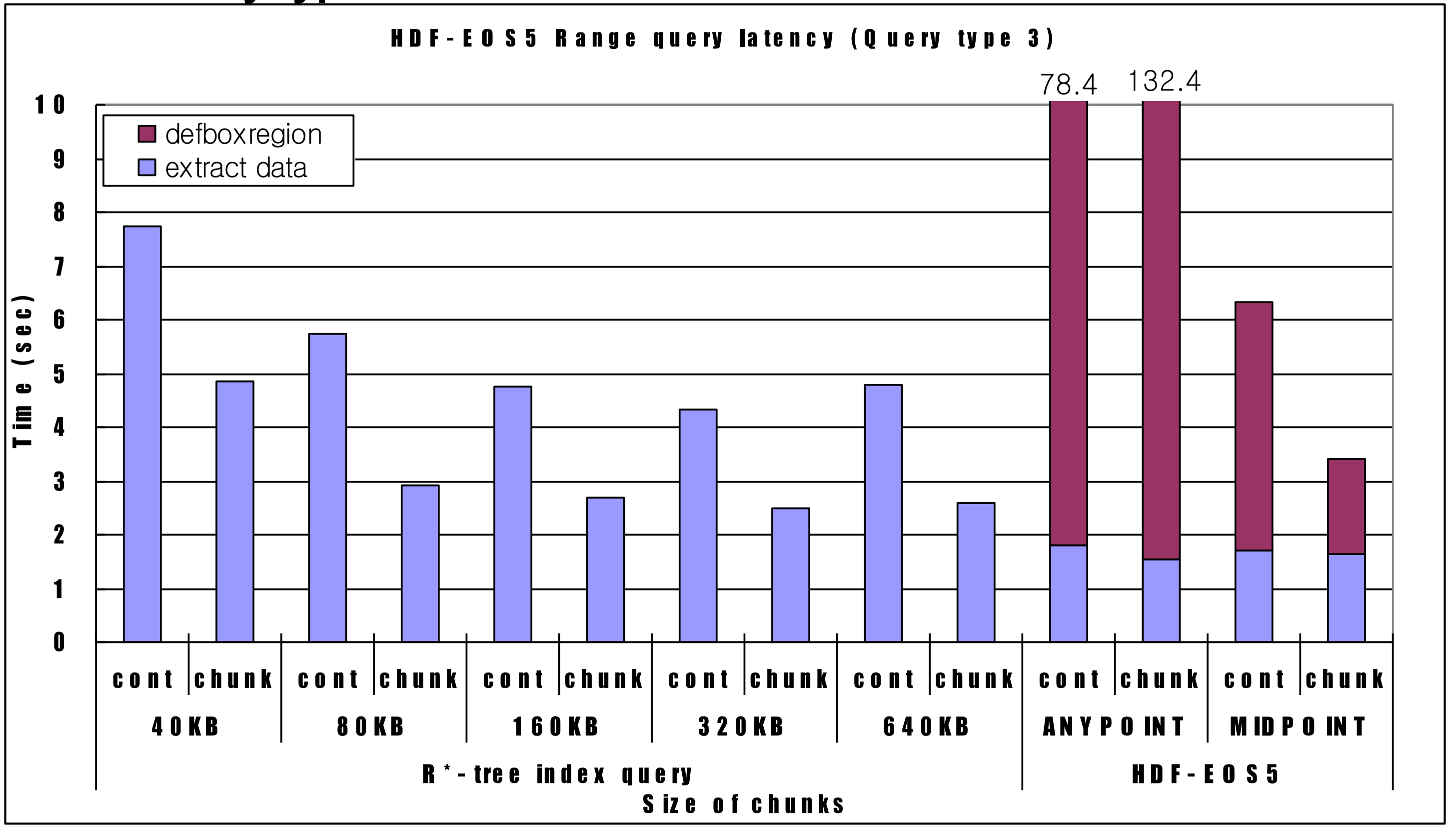


Range query performance (3)



Query type 3

Time to read a region with many rows and few columns



SH-Trees

Indexing Data Structures

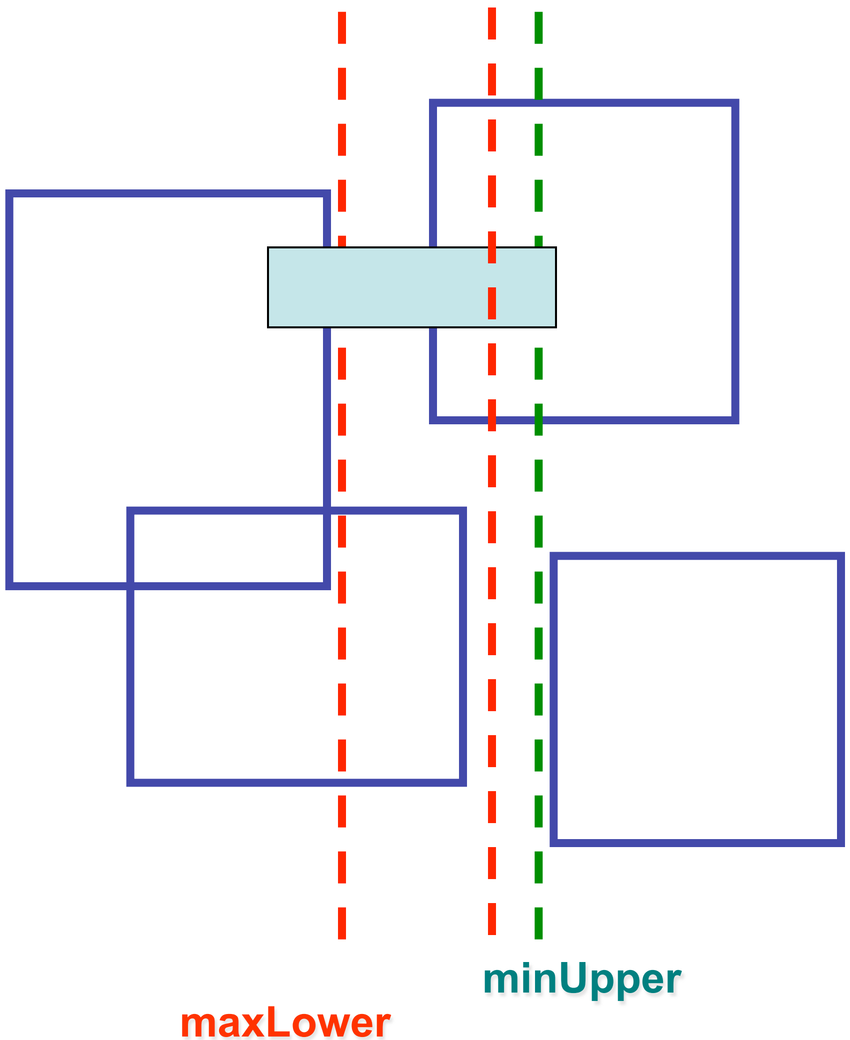
- Space partitioning method: point data
  - KDB-trees, Hybrid-trees, hB-trees, etc.
- Data partitioning method: non-point data (hyper-rectangles)
  - R-trees, R*-trees, X-trees, etc.
- Which indexing structure performs best for chunked datasets?
  - Spatial Hybrid Trees (SH-trees)

SH-Trees

- An extension of Spatial KD-tree and Hybrid-tree
- A disk-based space partitioning method for non-point data
  - Simple insertion algorithm -> fast insertion
  - Dimension independent -> no large fan-out -> fast search
  - allow (limited) overlap between partitions
- Good performance for both insertion and search
  - R-tree variants sacrifice insertion performance for faster searching
  - Better insertion & search performance for chunked datasets than R-tree extensions

Spatial Hybrid-trees (SSDBM 2004)

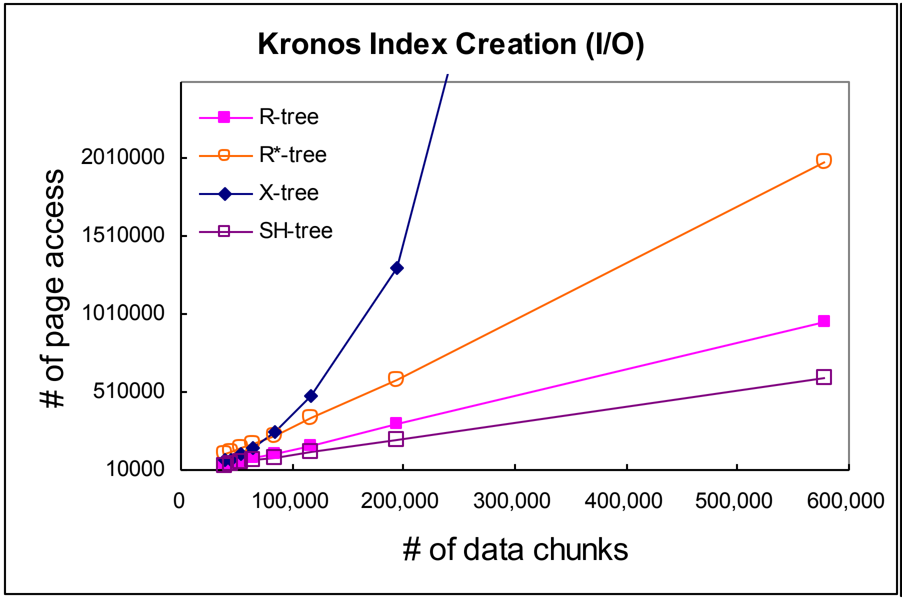
- Node Split Algorithm
  - Goal of node split is to minimize (maxLower – minUpper)
  - Iterate for each dimension to find minimum (maxLower – minUpper), and split that dimension
- Node Insertion Algorithm
  - Update one of split positions to include the newly inserted object

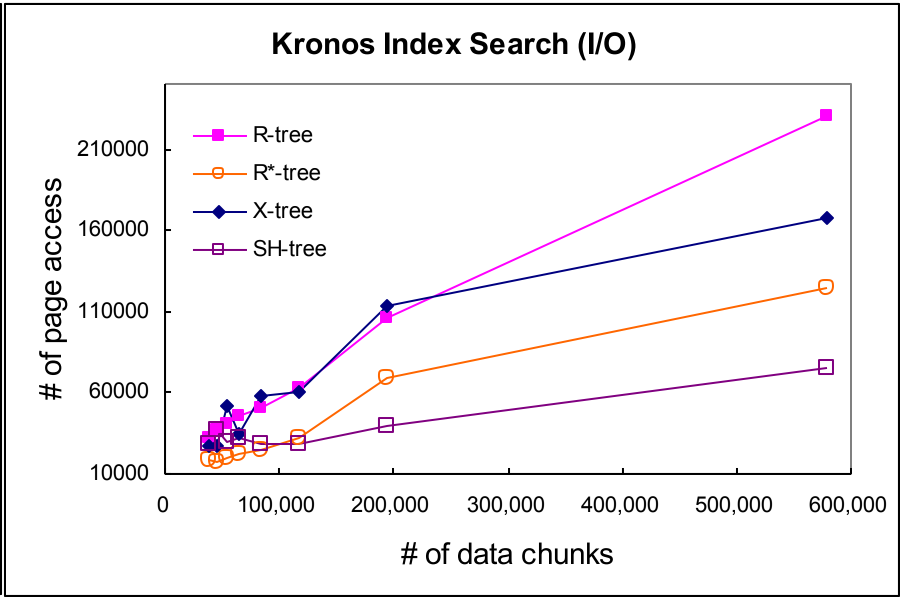


Performance Evaluation

- Platform
  - SunBlade 100 (500MHz Sparcv9, 256MB, 7200RPM IDE, 9ms seek time)
  - Turned off file cache
- Kronos Landsat Dataset
  - 3D AVHRR level 1B datasets (Latitude, Longitude, Time, 5 sensor values)
  - One month (Jan. 1992) ; 30GB
  - Workload generator (Customer Behavior Model Graph)
- Synthetic Dataset
  - Uniformly distributed 200,000 high dimensional hyper-cubes in the unit hyper-cube
  - Randomly generated queries
- Implementations
  - R-trees, R*-trees, and X-trees from **chorochronos.datastories.org**, with some minor modifications for fair comparison

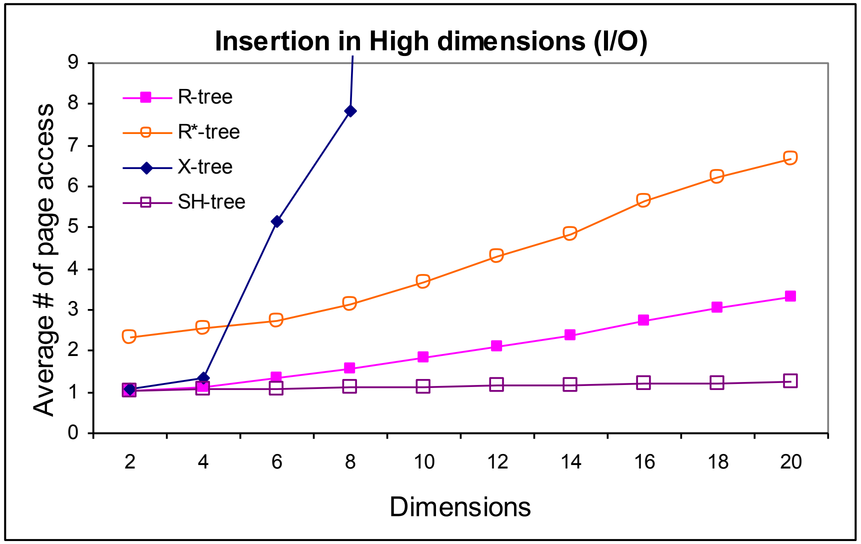
Performance on Kronos dataset

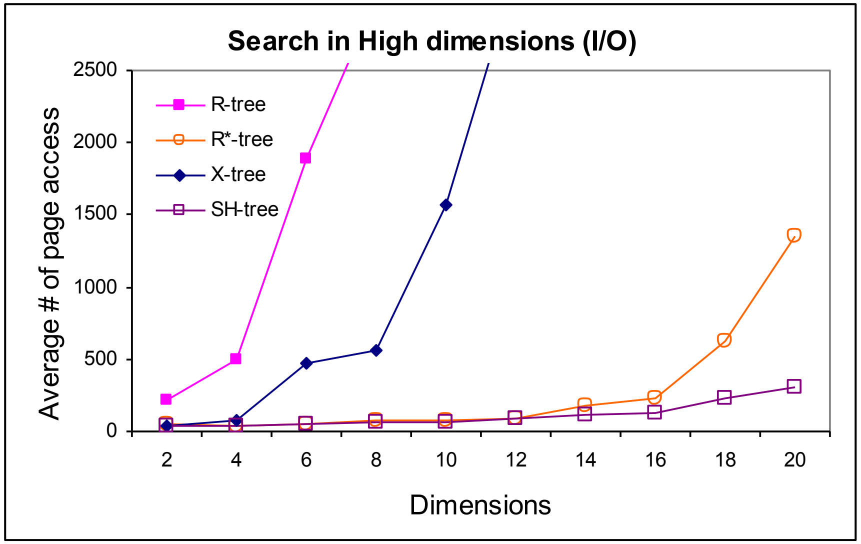




- Insertion:
  - I/O: SH-tree < R-tree < R*-tree < X-tree
  - Time: SH-tree < R-tree < R*-tree < X-tree
- Search (sum over 2,000 queries)
  - I/O : SH-tree < R*-tree < X-tree < R-tree
  - Time: SH-tree < X-tree < R*-tree < R-tree

Performance for Synthetic Dataset (High Dimensions)





- Insertion (200,000 hyper-cubes)
  - I/O : SH-tree < R-tree < R*-tree < X-tree
  - Time: R-tree <= SH-tree < R*-tree < X-tree
  - Size of X-tree root node is 667 pages
- Search (Average over 10,000 queries)
  - I/O : SH-tree < R*-tree < X-tree < R-tree
  - Time: SH-tree < X-tree < R*-tree < R-tree
  - X-tree search time is fast, but SH-tree is better
  - SH-tree performance is almost independent of # dimensions

Indexing Distributed Datasets

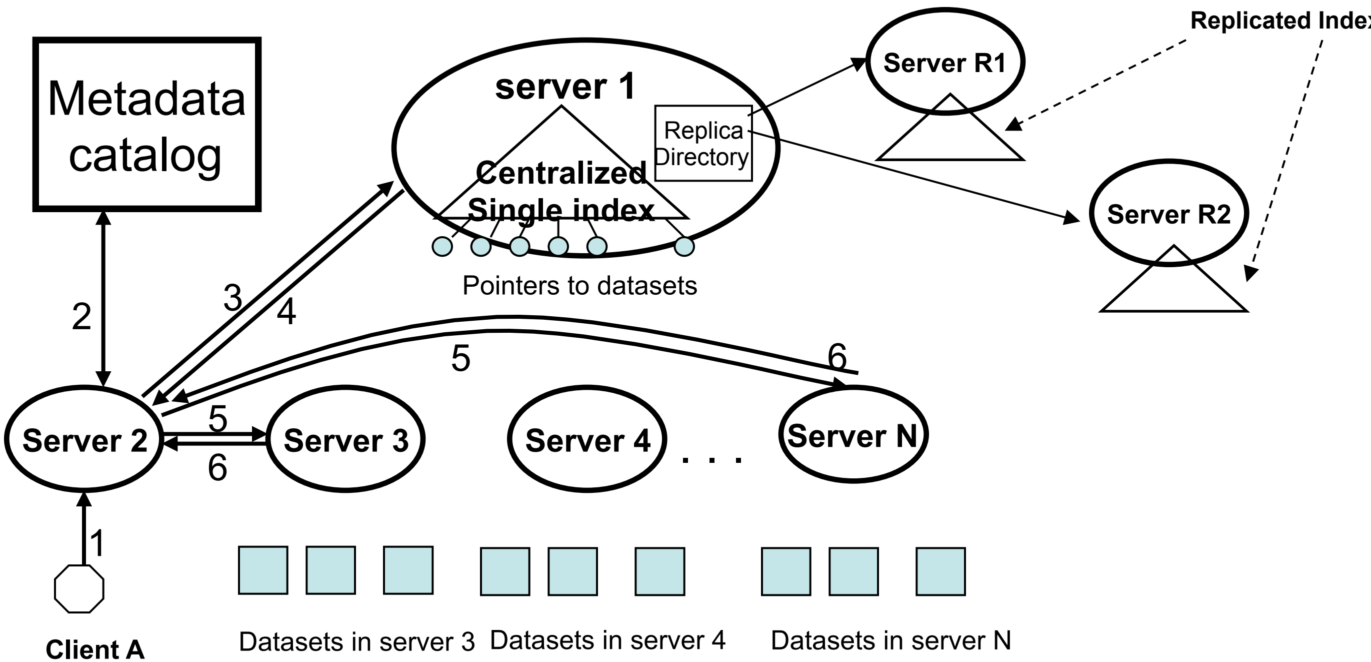
Distributed Spatial Indexing

- Motivation
  - Geographically distributed scientific datasets
  - Centralized index server likely to become a bottleneck
- Replicated Centralized Index
  - Replicate the whole index onto multiple servers
- Two-level Hierarchical Index
  - Each server maintains its own index
  - Top-level index server maintains the bounding boxes of root nodes of local indexes
- Decentralized Index
  - No centralized index server as in P2P systems
  - Distributed KD-trees: partial global index

Replication vs Hierarchical Index
(CCGrid 2005)

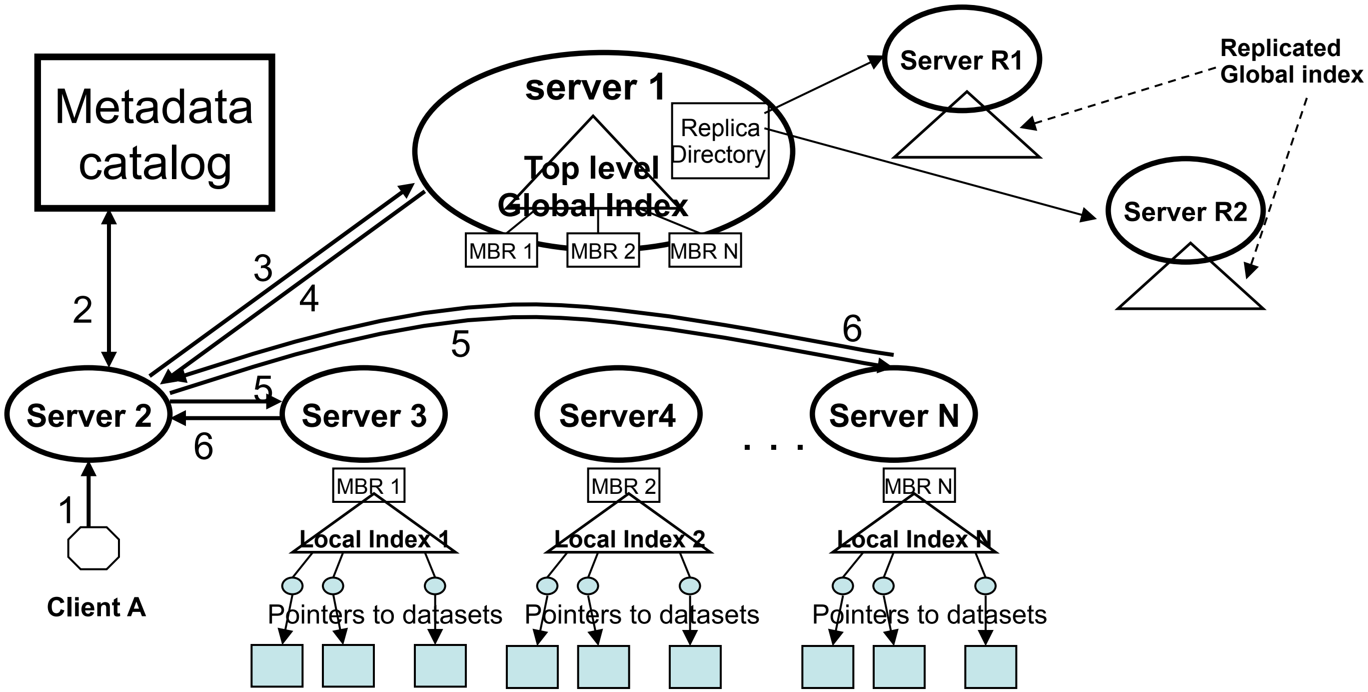
- Replicated Centralized index
  - Consistency problem
    - Insertion without replication is already expensive
    - Can guarantee correct results with inconsistent replicas
- Hierarchical Two-Level Index
  - More scalable than a single centralized index server
  - Still a potential performance bottleneck
- From the experiments and mathematical model
  - Observation 1: As the # of replicas increases, centralized index becomes faster than two-level index
  - Observation 2: As the # of servers increases, two-level indexing becomes faster than centralized index

Centralized Indexing



- Single server stores all the index nodes
  - Master R-trees
- Replication of centralized index
  - Copy the whole index onto multiple replica servers

Two-Level Hierarchical Indexing

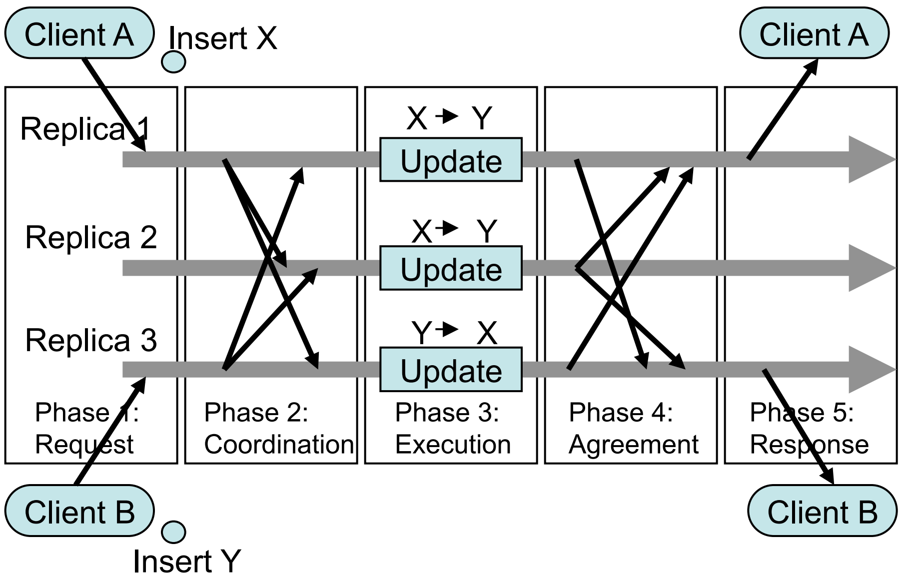


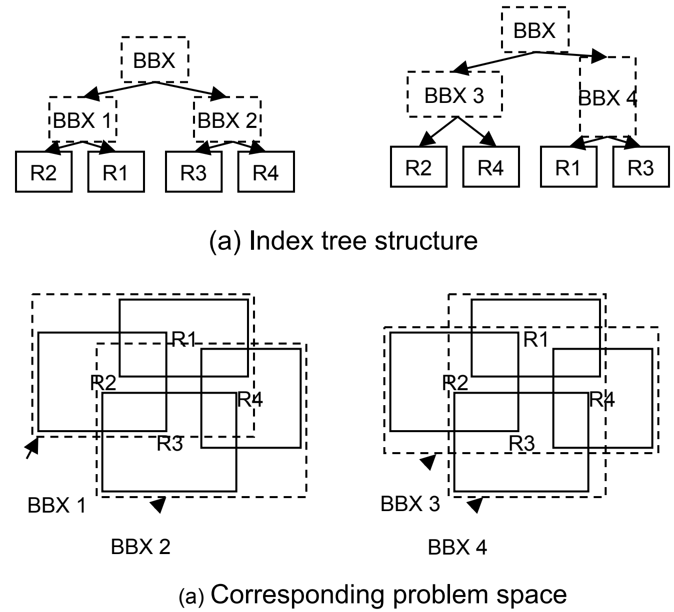
- Each data server has its own index for local data
- Small global index, better performance
- The global index in a centralized server stores MBR's of local indices
  - Master Client R-trees
- Still global index server is a potential bottleneck
- Replication of global index
  - No point in replicating local indices

Replication

- Why to Replicate
  - Better performance
  - Better availability
- Why Not to Replicate
  - Extra overhead
  - Deadlocks
  - Stale data
  - Reconciliation
- Replication for Multidimensional Index
  - No need for strict consistency among replicas
  - No write-after-write data dependency =>no deadlock
  - No need for reconciliation

Concurrency Control of Replicas



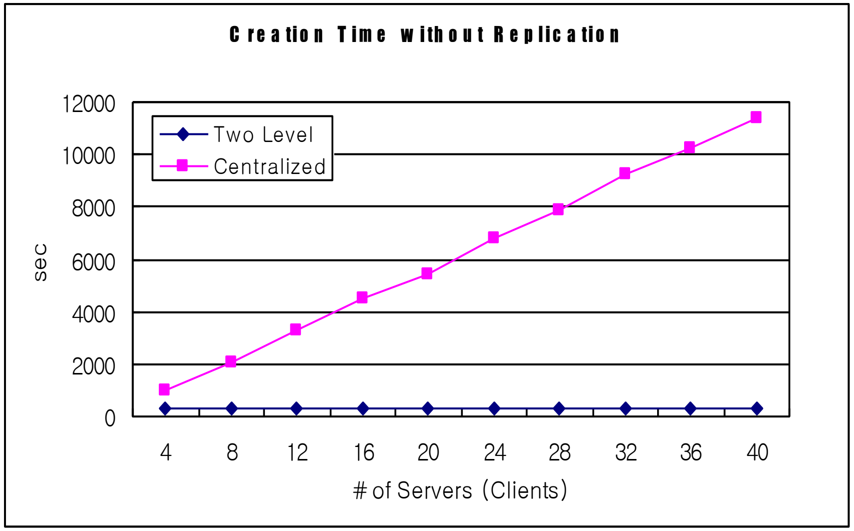


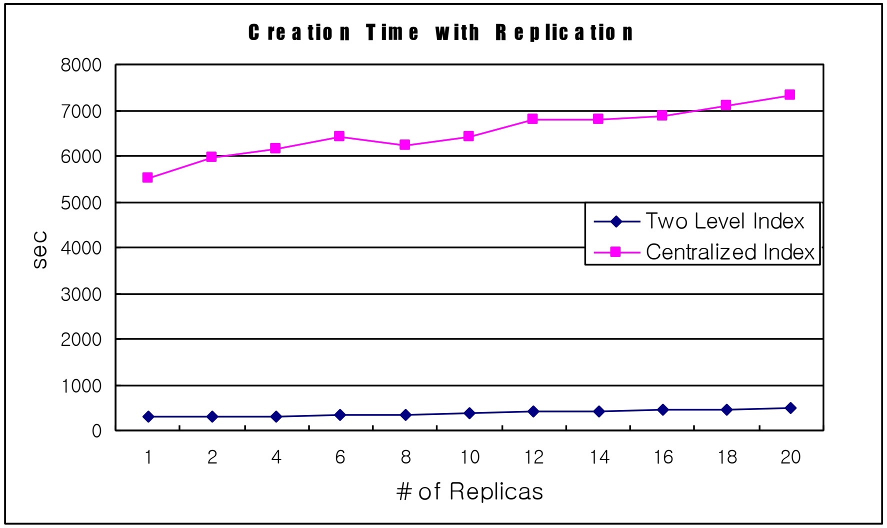
- Concurrent multiple insertions results in different tree structures.
- Nondeterministic internal structures in multidimensional index
  - Same data, different index
- Why strong consistency, if correct search result is guaranteed?

Experiments

- Storage Resource Broker (SRB)
  - Storage middleware for distributed environment
  - Implemented multidimensional indexing service on top of SRB as proxy functions
- Experimental Environment
  - 40 Linux nodes at U.Maryland (PentiumIII 650MHz, 100Mb/s Ethernet)
  - 20 Linux nodes at Ohio State University (PentiumIII 933MHz, 100MB/s Ethernet)
  - WAN: Internet2
- Datasets
  - AVHRR GAC level 1B satellite datasets
  - One month data (30GB)
  - Partitioned into 400,000 chunks
  - 10,000 chunks in each of 40 nodes
  - Query Workload (CBMG) : 2 clients per server, 100 queries per client

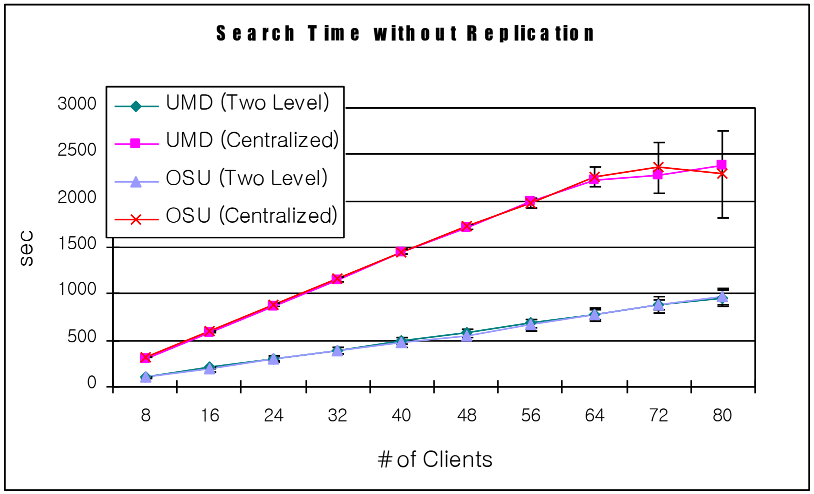
Experiments: Index Creation

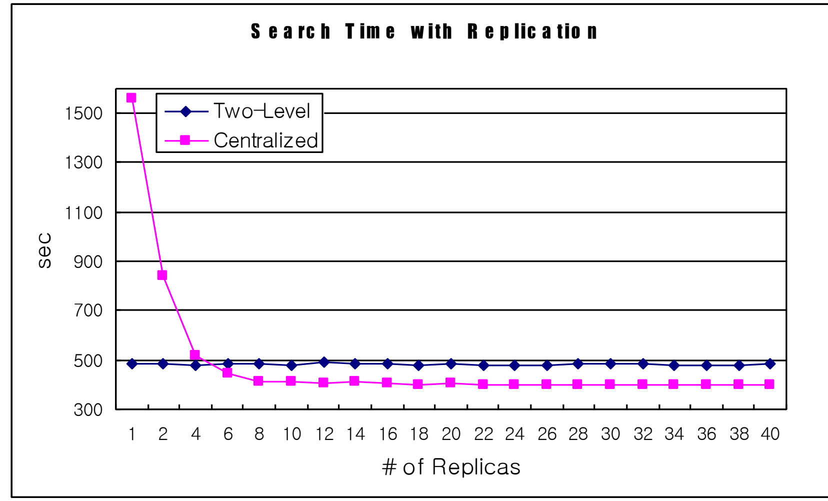




- Insertion performance of Two-Level index is superior
  - Most updates are done only in local index
  - Only when root bbx changes, access top-level index server
- Replication hurts insertion performance a little
  - 32% increase in centralized index (20 replicas)
  - 58% increase in two-level index, from a much lower starting point

Experiments: Index Search





- Non-replicated centralized index
  - Search cost is substantial compared to two-level index
  - suffers from resource contention.
- When centralized index is replicated, it becomes faster than replicated two-level index
  - Due to remote local index search for two-level index

Conclusions

- Spatial indexing of large datasets helps
  - both for searches and for updates
  - if use the right data structure – SH-trees
- Can effectively index distributed datasets
  - hierarchy and replication
- Have applied this to challenging data intensive applications
  - one example is multi-query optimization, to find cached aggregates in distributed query system
- Also have investigated decentralized indexing

Effective Indexing of Distributed Multidimensional Scientific Datasets

Alan Sussman
Beomseok Nam

UNIVERSITY OF MARYLAND

Department of Computer Science &
Institutute for Advanced Computer Studies

http://www.cs.umd.edu/projects/hpsl/chaos/ResearchAreas/gmil