Multi-dimensional Index on Hadoop Distributed File System

Haojun Liao Jizhong Han Jinyun Fang

{liaohaojun, hjz, fangjy}@ict.ac.cn

2010-7-15

中国科学院计算技术研究所

Institute of Computing Technology,Chinese Academy of Sciences

Contents

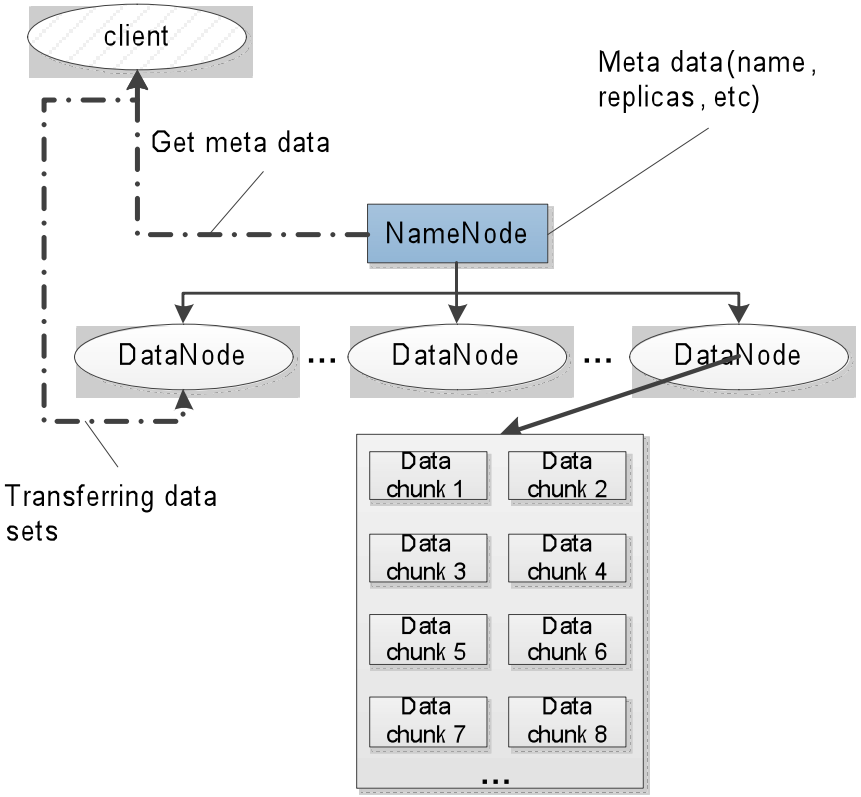
- Introduction
- Design Overview
- Implementation Details
- Performance Evaluation
- Conclusion & Future work

Introduction

- Data Volume
  - Data volumes increase rapidly from gigabytes to hundreds terabytes or even petabytes in recent years
- Scalability
  - Traditional databases designed monolithically are not easy to scale out
- Costs
  - Even if the distributed database is an option for large volumes data processing, it is far more expensive to afford
- Schema
  - Semi-structured and non-structured data emerging in modern applications cannot be handled efficiently by relational DB that are based on schema

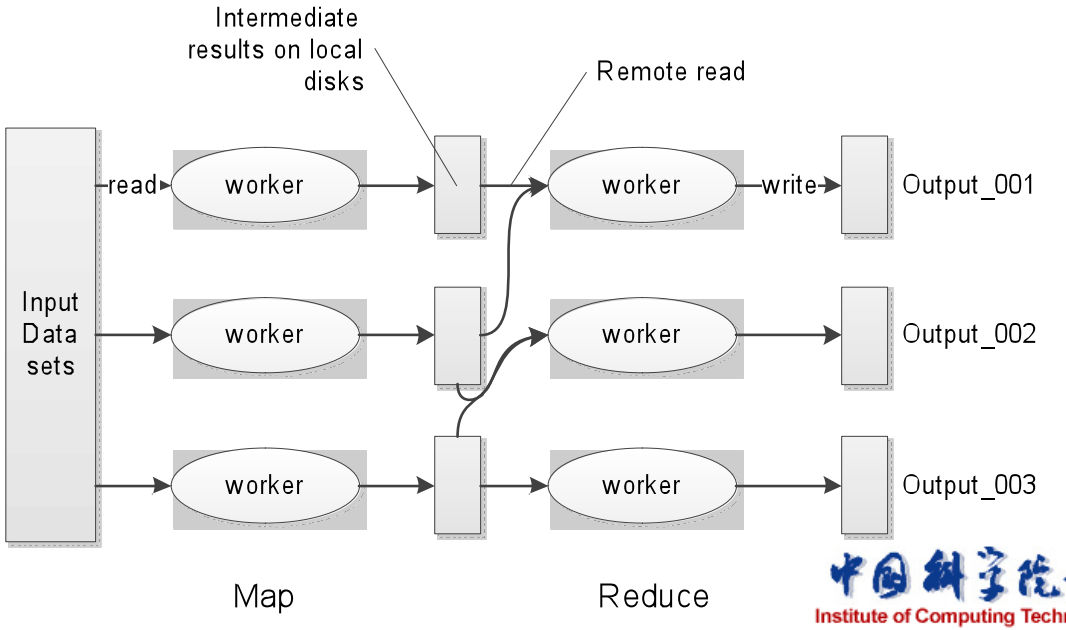
Introduction (cont'd)

- Hadoop
  - Hadoop Distributed File System
    - Master/Slave Architecture
    - One NameNode & several to thousands DataNodes
    - NameNode maintains meta-data, and DataNode serves data to client
    - File in HDFS is partitioned into chunks, and several replicas exist for each data chunk



Introduction (cont'd)

- MapReduce
  - Consists of Map & Reduce phases
  - Intermediate result are materialized in local disks
  - Reduce workers need to remotely load data for computing purpose
  - Processing in Reduce follows shared-nothing fashion
  - Integrate user-defined Map/Reduce functions into framework



Introduction (cont'd)

- MapReduce Debates
  - MapReduce is not novel at all and misses many features that are included in modern DBMS
    - Indexing: B-tree or Hashing
    - Transactions
    - Support results-oriented language, like SQL
  - The advantage of MapReduce is that it can distribute data processing across a network of commodity hardware
  - Not tuned to be high performance for specific tasks, still can achieve high performance by being deployed on a large number of nodes.

Introduction (cont'd)

- Enhancements for Hadoop in HIVE
  - Support results-oriented language: SQL
  - Provide enterprise standard application programming interface: JDBC, ODBC
  - Manage structured data based on HDFS, supporting primitive types: integer, float, Boolean, etc.
  - Designed for generalized data processing, combining with query optimization strategies
  - Use database for the management of meta-data of files that are stored in HDFS for efficient retrieval

Introduction (cont'd)

- Hybrids of MapReduce & DBMS in HadoopDB
  - Combine the scalability of MapReduce to DBMS
  - MapReduce is responsible for coordination of independent databases
  - Equip with standard interface, like SQL, JDBC, brought by DB
  - Structured data processing is delegated to underlying DBMS
  - Additional work is required to answer complex queries that underlying DBMS cannot support, like processing of multi-dimensional data
  - Query optimization constrained by underlying DBMS as well as the MapReduce framework

Introduction (cont'd)

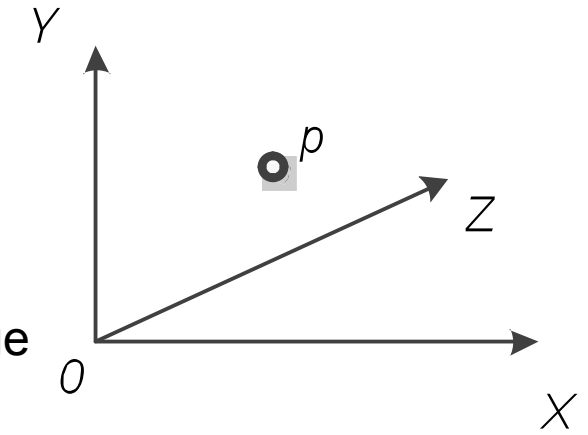
- Index structure in HDFS makes life easy
  - Answering queries by using access methods can be more efficient than by using sequential scanning
  - Already proposed efficient techniques based on index for answering queries are all available with minor modification
  - Efficient supporting complex query types based on index structure is achieved by exploiting filter-refinement paradigm
  - Query processing using index can be integrated with MapReduce framework towards query efficiency
  - Support new data types and related queries, like multi-dimensional data and spatial queries

Design Overview

- Data types in real world
  - One dimensional data (alphanumeric data)

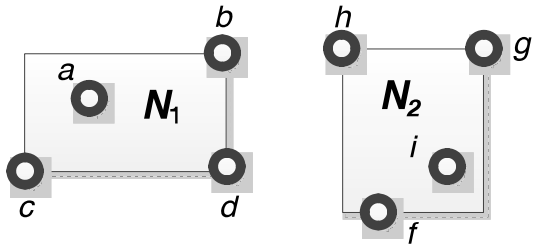
Name: Mike Address : Haidian District, Beijing, P.R. China

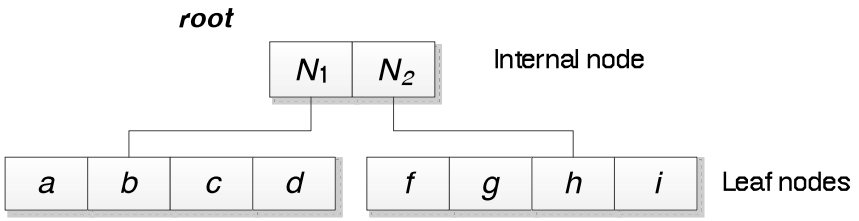
  - Multi-dimensional data
    - Spatial data is a typical type of multi-dimensional data
    - Others types include audio data, video data, and image
    - Multi-dimensional data are usually more complex, not single valued, and large in volume



Design Overview (cont'd)

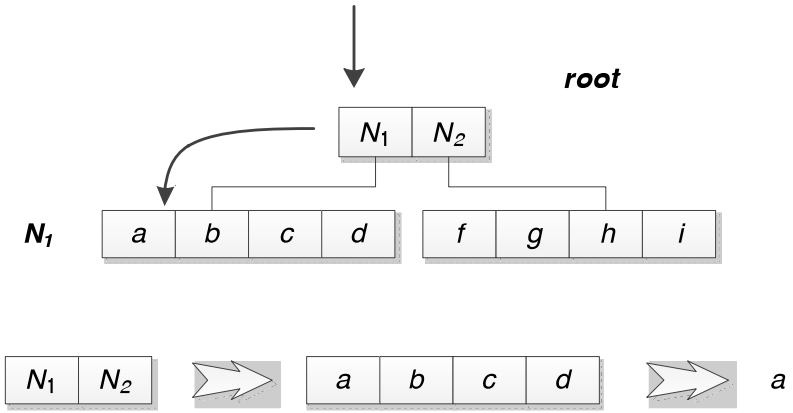
- Tree-like (hierarchical) index structure
  - Typical example of tree-like index structure is the B-tree
  - Typical example of multi-dimensional index is R-tree, an multi-dimensional extension of B-tree
  - Block-based secondary index structure
  - Index nodes are of the same size in a index





Design Overview (cont'd)

- Query processing based on index
  - At least *N* index node needs to be loaded into memory, where *N* is the height of index
  - The number of index node that will be visited during query is in proportion to the final results sets.



Example of Point query

1. Load root node, and check which entry contains the query point in geometry sense

2. Load leaf node N_1, referenced by entry N_1 in root node, and check entries in node N_1 one-by-one to determine the final result, the object a

Design Overview (cont'd)

- The gap between HDFS and the requirements of index

Characteristics of HDFS

1. Data are transferred in the form of data packet
2. Data are pushed to client from data node, and efficient for sequential read, not for random access
3. All files are partitioned sequentially by using range partition method
4. All data chunks are treated equally.

Requirements of Index

1. Efficient block-based access pattern
2. Index nodes need to be loaded on-demand efficiently
3. Efficient random access
4. Low data communication overhead

Implementation Details

- Techniques to address problems

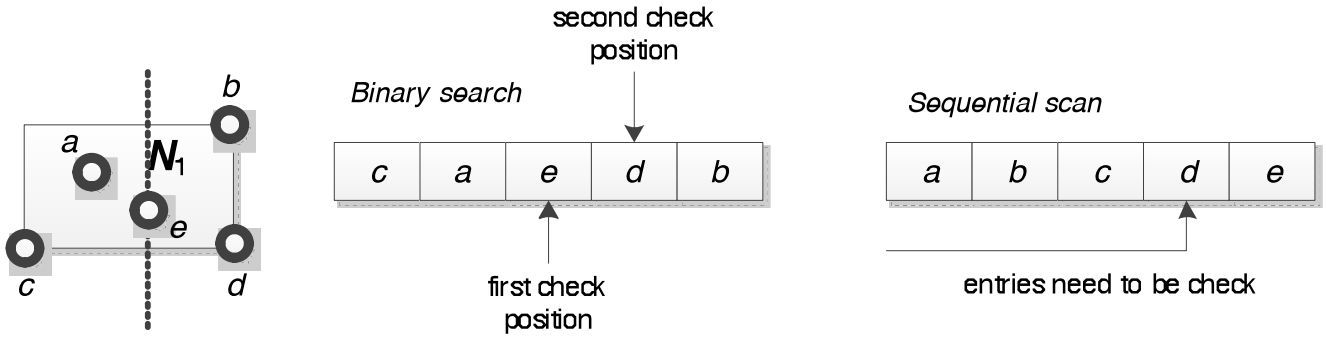
Characteristics of HDFS	Our solution
Data transferred in data packet	Tuning node size and ordering entries in index nodes
Push model	New data transferring model
File partitioned	Re-organizing index nodes on disk
Data chunks are equally treat	Buffer strategy for upper level

Implementation Details (cont'd)

- Tuning index node size
  - Determine the most effective node size to facilitate I/O performance
    - Small index node incurs more data transmission times
    - Large index node causes the transmission of unnecessary data
    - Large node size decrease the discrimination power of index by using more CPU efforts to filter the result.
  - The size of index node should be aligned to the size of data packet, the minimum unite for data transferring in HDFS
  - Avoid the transmitting of the unnecessary data

Implementation Details (cont'd)

- Order the entries in each node
  - Ordered entries can speed up the in-memory search procedure by reducing computing needs

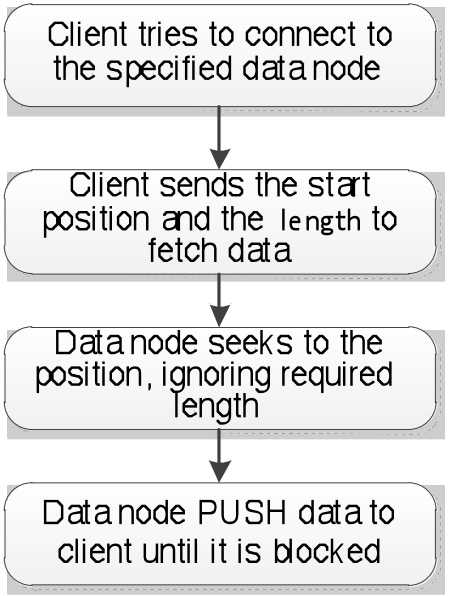


Example of point query

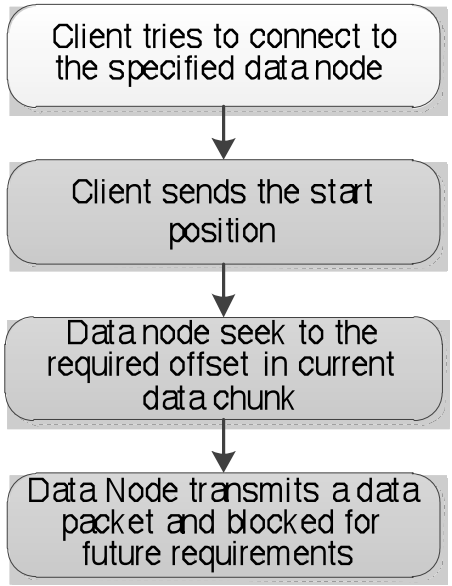
  - Save the sort operations that are necessary for many query processing:
    - spatial join – I/O and CPU-intensive query

Implementation Details (cont'd)

- New data transferring model in HDFS
  - The main difference with PUSH model is that Datanode is blocked after transmitting one data packet.



Original PUSH model



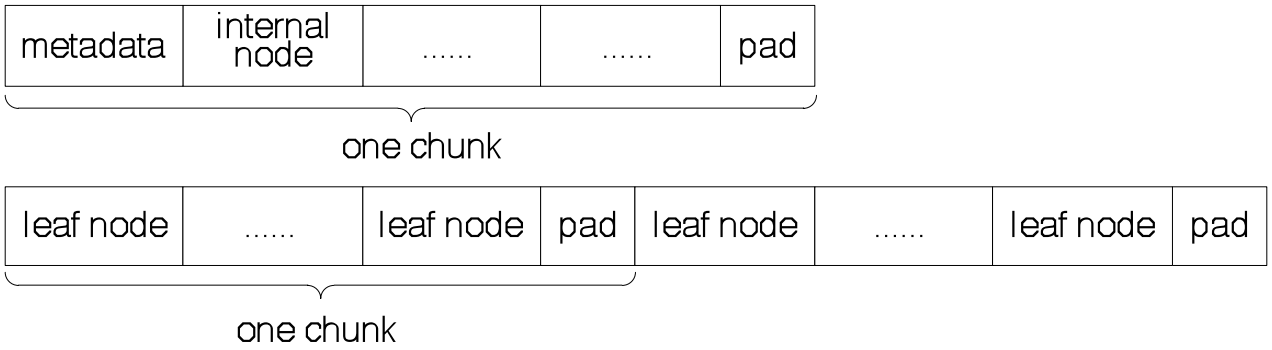
Our transfer model

Implementation Details (cont'd)

- New data transferring model in HDFS
  - Datanode is blocked in favor of the random access where the client might not need the sequential data packets.
  - Inferior in sequential data transferring compared with push model
  - Superior in random access of data sets and incurs no redundant data packets that are transmitted to client but useless.
  - The size of data packet still affects the I/O performance over network as happens in PUSH model

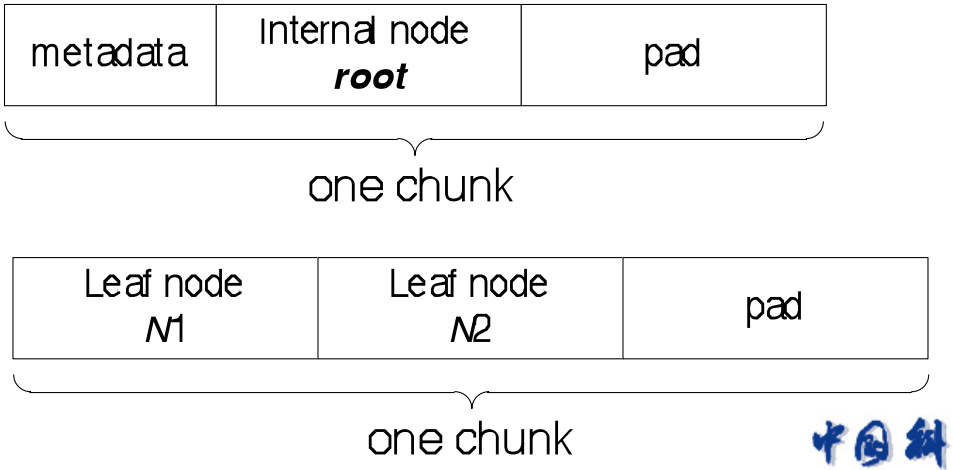
Implementation Details (cont'd)

- Index structure in HDFS
  - Meta-data locates in the front of file, accounting for one kilobytes disk space
  - Internal nodes follows the meta-data, clustered in the first chunk.
  - Remaining space in the first chunk except internal nodes and meta-data are left for future use



Implementation Details (cont'd)

- Index structure in HDFS
  - Leaf nodes are clustered according to location proximity
  - No leaf nodes cross two data chunk to avoid visiting one index leaf node incurring two counts of TCP connection
  - The example of index structure of aforementioned index is illustrated:



Implementation Details (cont'd)

- Buffer strategy
  - Properties comparison of both internal and leaf nodes

Properties of internal nodes	Properties of leaf nodes
Relatively small	Relatively large
Access frequently	Visited on demand
Clustered in one data chunk	Distributed across many nodes

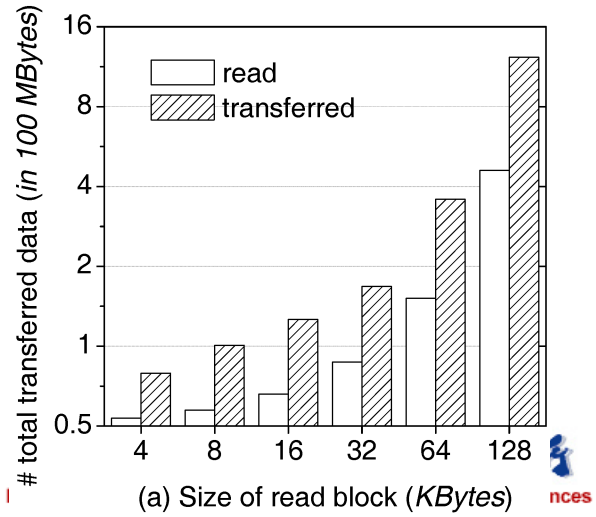
Implementation Details (cont'd)

- Buffer strategy
  - Internal nodes are pinned in buffer once loaded for future node access
  - Leaf nodes are allocated limited number of buffer pages averagely distributed in each data nodes that are managed by LRU policy
- Data transferring procedure
  - Once the data request emerges, data node check its buffer first
  - If the required data packet is hold in the buffer, it is sent to client
  - Otherwise, disk access is invoked.

Performance Evaluation

- Datasets
  - CAR contains 2,249,727 road segments extracted from Tiger/Line datasets;
  - HYD contains 40,995,718 line segments representing rivers of China
  - TLK contains up to 157,425,887 points.

- Transferring overhead
  - Redundant data transferred by using PUSH model
  - Vary the size of read block and the data packet is set 64*K*



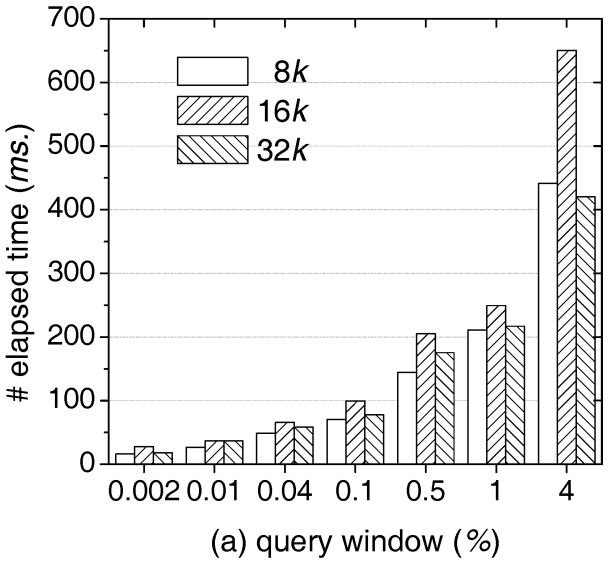
Performance Evaluation (cont'd)

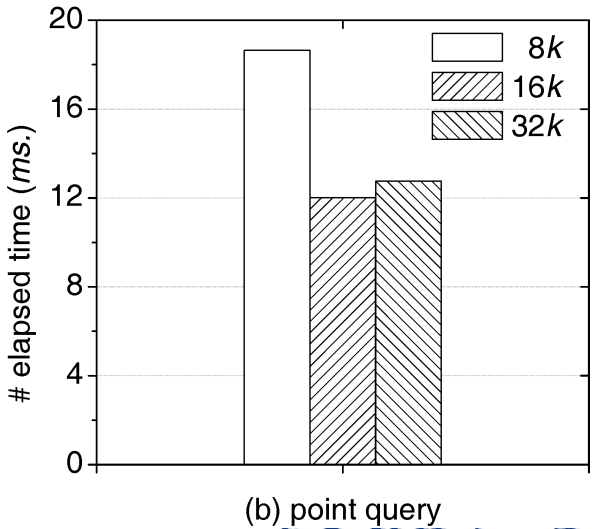
- The comparison between *PUSH* model and our new data transferring model

	Data usage	Sequential read	Random read	Localized random read
PUSH model		√		
Our transfer model	√		√	√

Performance Evaluation (cont'd)

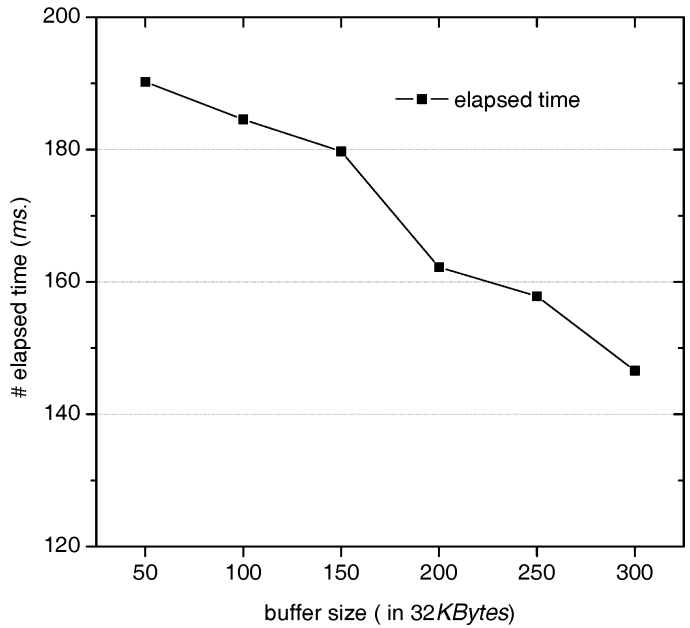
- Experimental results of node size effects
  - Investigate the impact of node size to both range query and point query by varying node size in query processing
  - In the range query, query windows ranges significantly (0.002% to 4% of total space), and we measure the query response time





Performance Evaluation (cont'd)

- Buffer effects
  - Vary the buffer size to evaluate the effects of buffer on query performance, whereas other parameters are kept constant
  - The query used here is the range query with query window 1% of TLK data set



Conclusion & Future work

- Conclusion
  - We propose a method for organizing hierarchical structures applied to both B-tree and R-tree on HDFS
  - We investigate several systematic parameters like node size, index distribution, buffer, and query processing techniques
  - Data transfer protocol specified for block-wise random reads integrate with HDFS
- Future work
  - Investigate the problem of combination of MapReduce and index structure.
  - Explore efficient multi-dimensional data distribution strategy according to index structure to further enhance the I/O performance

Questions?

Thank You !

中国科学院计算技术研究所
Institute of Computing Technology,Chinese Academy of Sciences