Paging B-Trees for Distributed Environment

Shogo Ogura Takao Miura

HOSEI University

Contents

Goals

- Parallel Processing
  – To Distribute B-tree
- Data Migration
  – Well-Balancing of Data Distribution
- No Duplicate
  – No Synchronization

Related works (1)

- Traditional Approaches
  - Range-Specified Distribution
    - Poor Balance of Data Distribution
  - Hash Function
    - Hard to Perform Range-Query

Related works (2)

- B-Tree with Page Distribution
  - No Capability of Delete Operation
- Fat B-tree
  - Much Amount Synchronization at Update
    - Upper nodes are Duplicated

- In All these Techniques
  - Data could be Migrated but not Dynamically

Our Approach

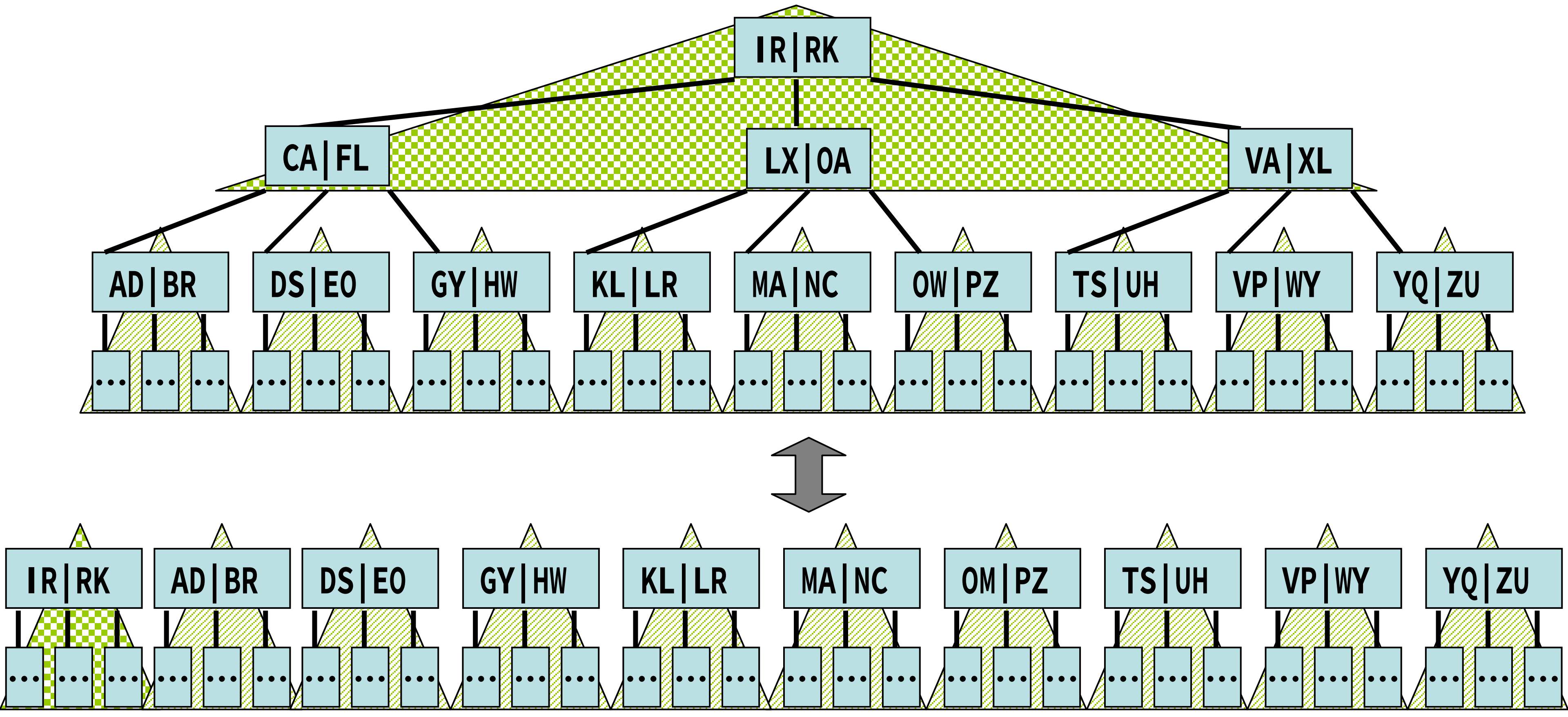
- Major Advantages
  - No Duplicate node
    - No Additional Update Overhead
  - Dynamic Data Migration
    - Independently of whole B-tree Structure
  - Range Query
    - In a Straightforward way
  - Many Chance of Optimization
    - Such as Cache Buffering

B-tree Paging

- Dividing tree into Several sub-trees
  - Processing in a Parallel manner
- Managing sub-tree at Any Hosts
  - By Specifying Additional Keys
    - Independent of Original (Local) Keys

Structuring B-Tree Pages

- Dividing depth 4 into depth 2
- Managed Independently

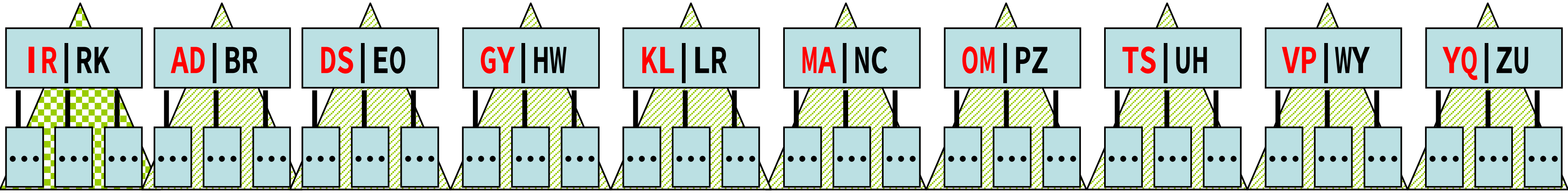


Managing Pages

- Virtualizing B-tree
  - Managing Correspondence
    - Logical Key
      - Root Key Value
    - Physical Key
      - Host Name
      - Position in the Host

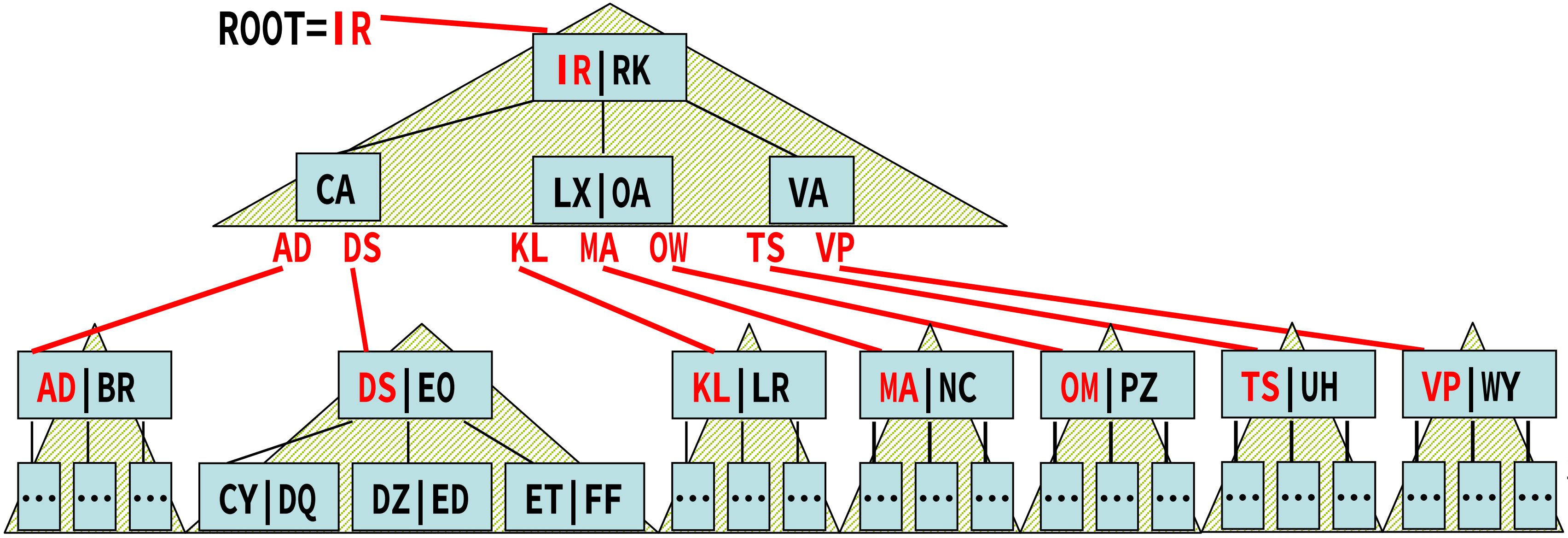
Page Map

Logical Key	Physical Key
AD	(Host-A, 1)
DS	(Host-A, 2)
GY	(Host-D, 2)
IR	(Host-C, 1)
KL	(Host-B, 1)
MA	(Host-B, 2)
OW	(Host-A, 3)
TS	(Host-D, 1)
VP	(Host-C, 2)
YQ	(Host-B, 3)



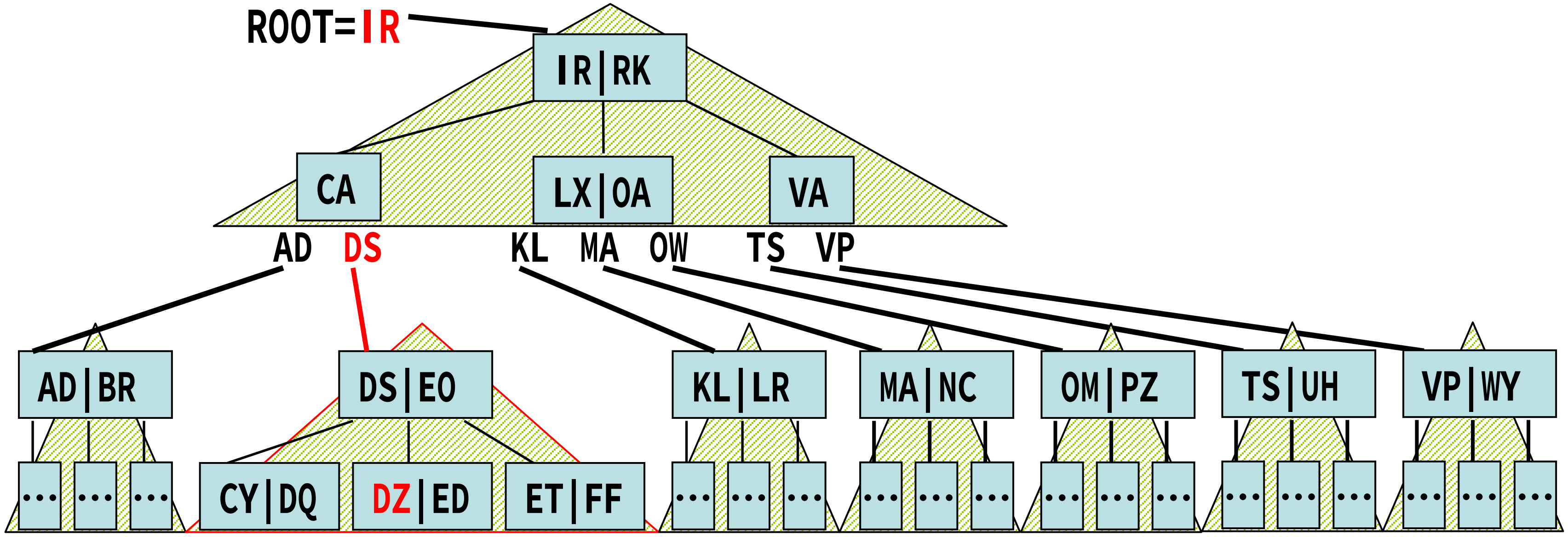
Querying B-Tree

- Crossing to a Lower sub-tree
  - Logical Key
    - Identify pages
    - Served as Pointers to Obtain Desired Data
    - Stored in each Leaf node of Upper sub-tree



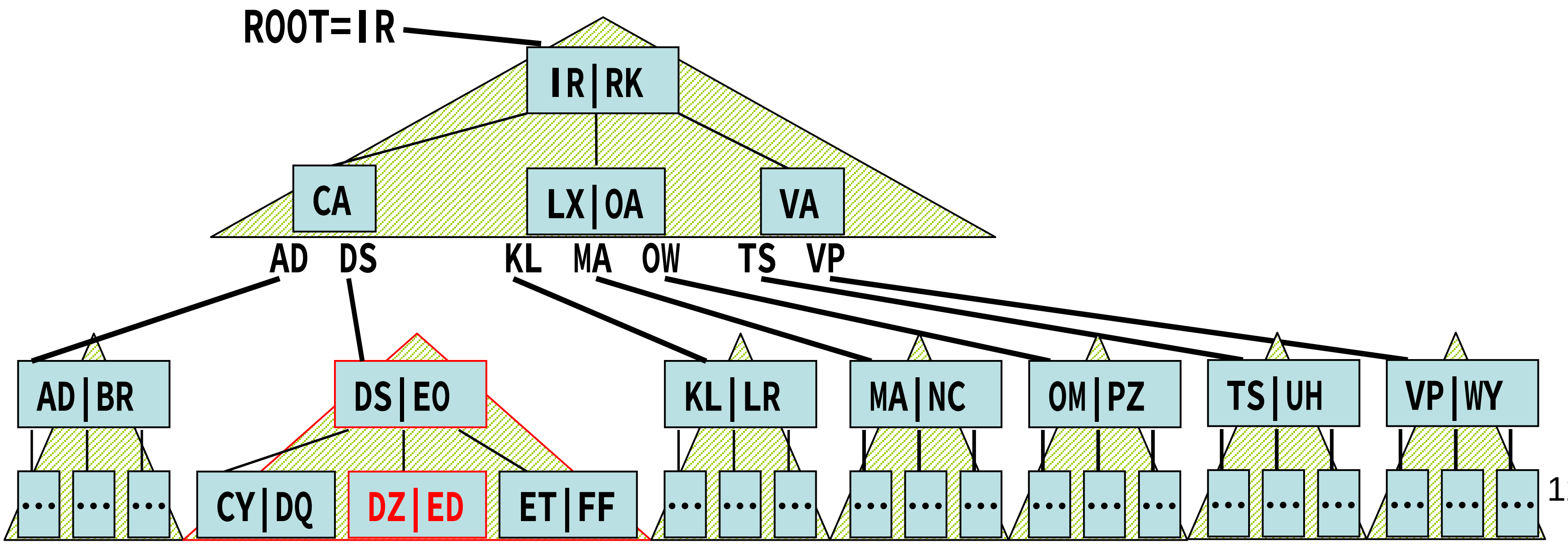
To Find a value “DZ”

- Examining a key “IR” of the Root at Page Map
- To Find a key “DS” in the Root sub-tree
  - Pointing a Lower sub-tree that Contain “DZ”
- Obtaining a Position of the Lower sub-tree
- Finding “DZ” in the Lower sub-tree



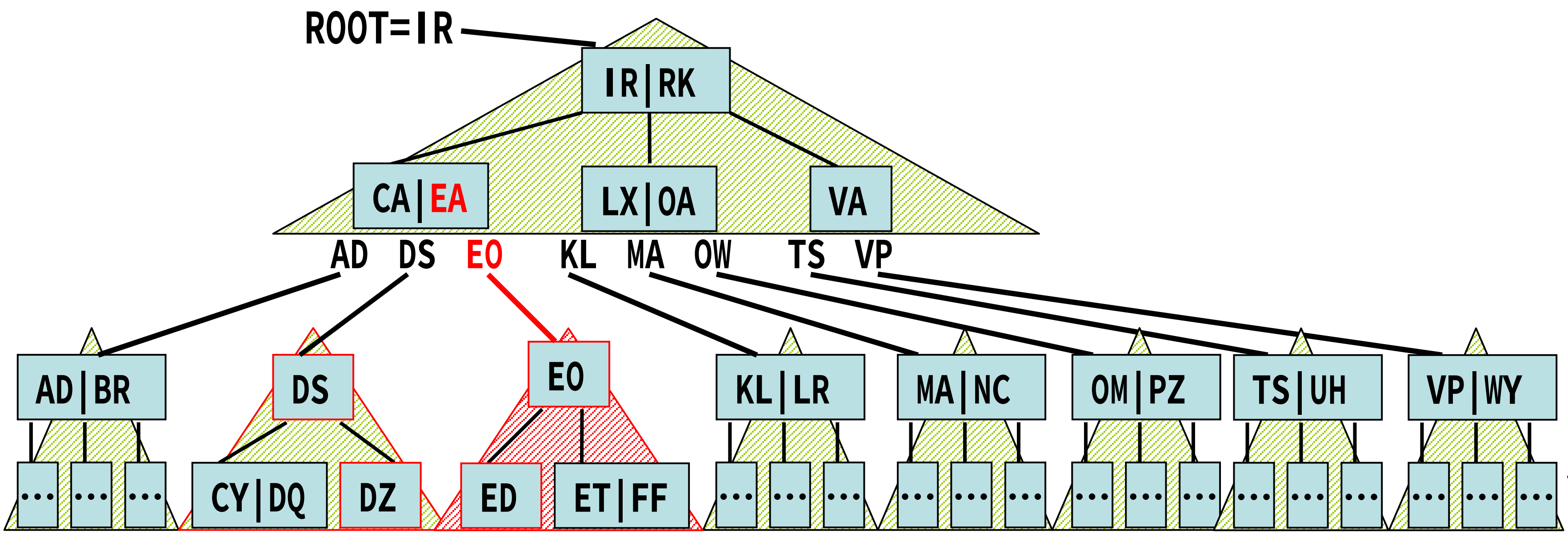
To Add "EA" into B-tree (1)

- Order : 1
- Exploring a position to be Inserted for "EA"
  - Between "DZ" and "ED"
- Splitting at this node
  - Root node of the sub-tree



To Add "EA" into B-tree (2)

- Splitting at Root Node of Sub-tree
  - New Sub-tree with root node "EO"
  - "EA" with the pointer to "EO"
    - Inserted toward Root sub-tree



Locating sub-tree

- To Transfer No Data among hosts
  - New sub-tree into Same host
- Any sub-tree can be moved Freely among Any hosts
  - Keeping the Logical B-tree Structure
    - By Taking the Correspondence of keys

Data Migration

- According to Some Rules
  - Locating into Same Cluster
    - Page containing many “Tokyo*” keys
    - Page containing many “Kanagawa*” keys
    - Since both prefectures are Close in Japan

- Well-Balance of Data Distribution
  - By Moving pages among Any hosts

Implementation

- Server/Client Architecture
  - One Administrative Sever
    - Managing Page Map
    - Asking Clients of Process Request in Parallel
  - Several Clients
    - Processing B-tree

Distributed Processing

- Sarver
  - Send (*command, physical-key, data*);
  - Receive (*status, data*);
  - …next Processing

- Client
  - Receive (*command, physical-key, data*)
  - *Command* (*physical-key*, *data*){
    - …Processing
    - Send (status, data);

    }

Experiment

- To Evaluate B-tree Paging as a Whole
  - Several Costs
    - Communication, Synchronization
      - For Parallel Processing
    - <u>B-tree Paging</u>, Data Migration
      - Fundamental Framework of Parallel Environment
  - Executing B-tree Paging in One host
    - Comparing with conventional B-tree

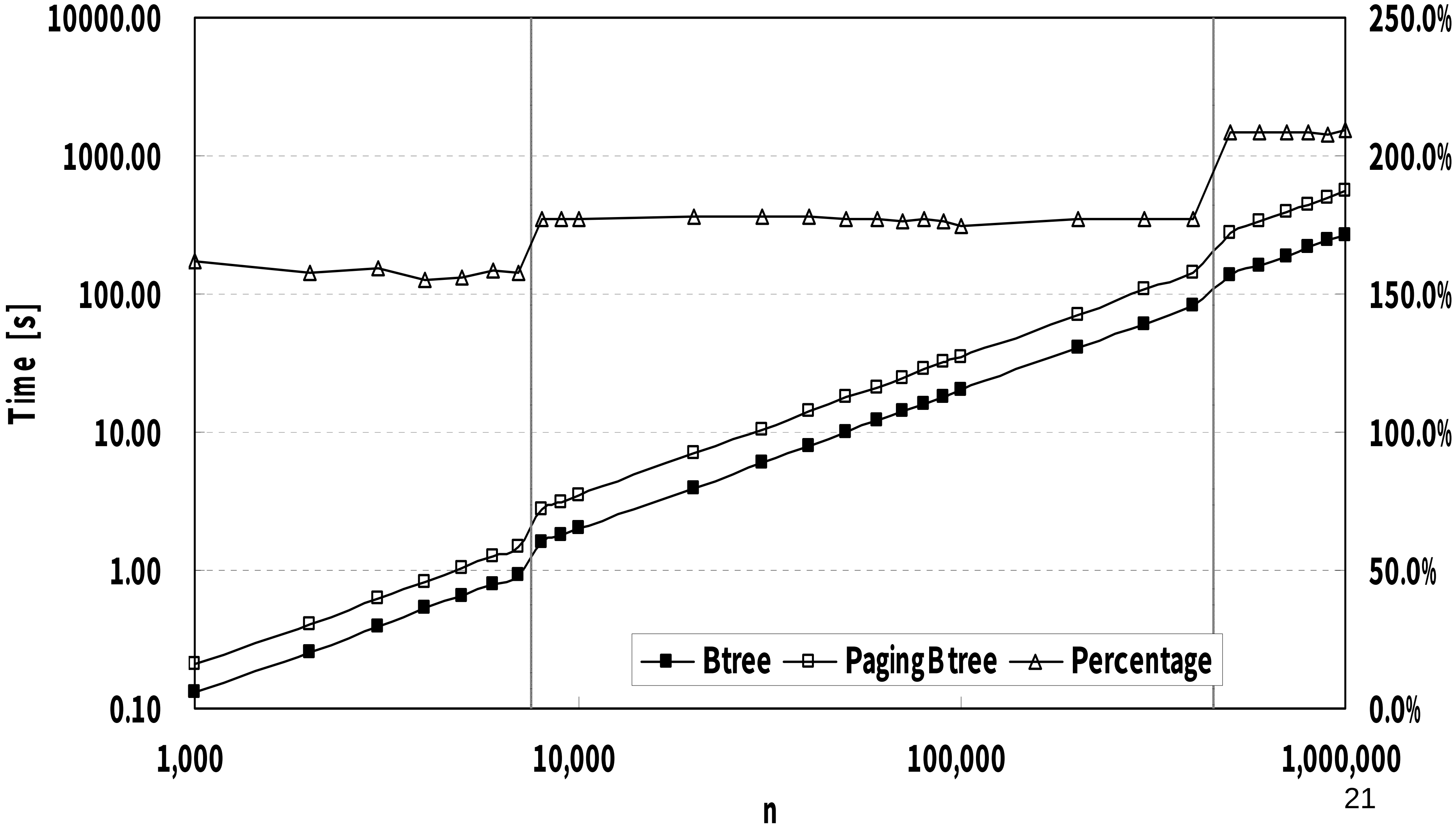
Experimental Environment

- FreeBSD4.1, PentiumIII(1GHz), IDE Ultra ATA-100 Hard Disk
- B-tree
  - order:50, data entry/pointer:4byte
- B-tree Page
  - depth:2
- Page Map
  - In a Memory in a form of B-tree

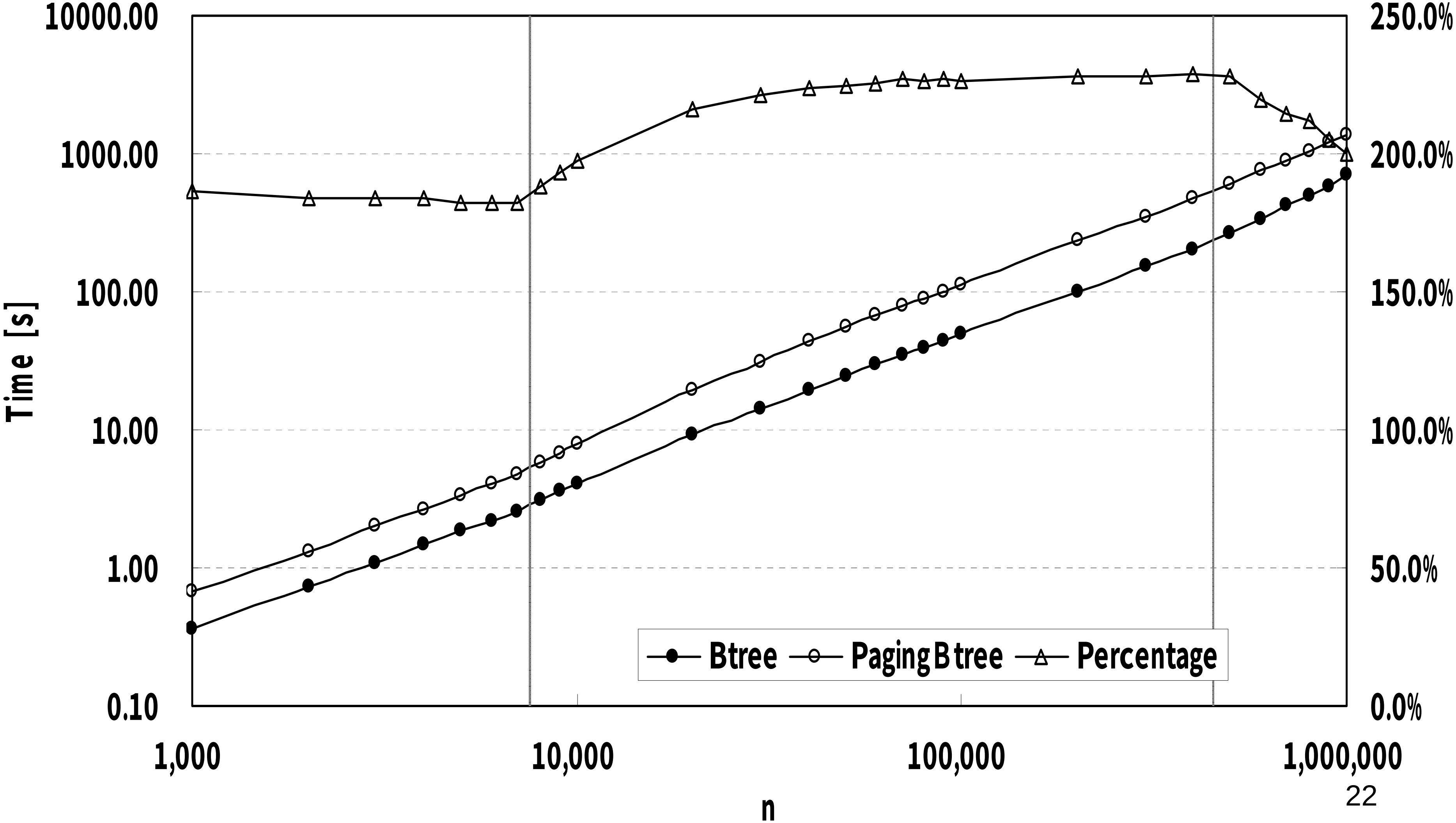
Experiment

- 10^6 Unique Random integers in a Array
- Adding first *n* integers into B-tree/B-tree Paging
  - $10^3 \leqq n \leqq 10^6$
- Querying B-tree/B-tree Paging

Result (Querying)



Result (Insertion)



Discussion

- No own Cache Technique
  - B-tree/B-tree Paging
- UNIX System Cache effect to I/O Operation
  - Impossible to Control Directly
    - Page Map, several Sub-tree are affected

Discussion (Querying)

- Depending on a Number of I/O
  - Path from Root to any Leaf
    - Including Page map

$$\frac{(\text{Path in Btree Paging})}{(\text{Path in Btree})} = \frac{(\text{Depth of Btree}) + (\text{Layer of Subtree}) \times (\text{Depth of Page Map})}{(\text{Depth of Btree})}$$

  - Depth of Page Map
    - Same Depth of the Upper sub-tree
      - Same number of sub-tree

Discussion (Querying)

Depth of B-tree	Layer of Sub-tree	Depth of Page Map	Expect Ratio	Experimental Result
2	1	1	150%	158%
3	2	1	166%	177%
4	2	2	200%	209%

- Overhead caused by B-tree Paging : 5%
- Page Map should be Managed in Memory
  - Few Overhead

Discussion (Inserting)

- Modifying Page Map
- Overhead caused by Splitting sub-tree : ~50%

Layer of sub-tree	Overhead	Splitting Sub-tree
1	80%	without Splitting
2	~130%	with Splitting

- In Parallel Processing
  - Better Efficiency
    - No Duplicate Overhead, No Synchronization Overhead

Conclusion

- B-tree Paging for Parallel Processing
  - Examining the Overhead of the Paging Mechanism
- Future Works
  - Direct Access to sub-trees
  - Effective Cache mechanism
  - Parallel Data Manipulation

Comparing with Fat B-tree

- No Duplicate
  - Synchronization
    - Update, System Crash
    - On the Internet
- Querying
  - Slightly Inferior
- Simple Architecture

Concentration of Load

- On a Upper sub-tree
  – Processing in Cash

Granule

- Freely Data Migration
- Optimization
  - – Same Host
    - A page with Many amount of data
    - A page with Few amount of data