Paging B-Trees for Distributed Environment

Shogo Ogura  Takao Miura
HOSEI University
Contents

1. Goals
2. B-tree Paging
3. Experimental Results
4. Conclusion
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

<table>
<thead>
<tr>
<th>Logical Key</th>
<th>Physical Key</th>
</tr>
</thead>
<tbody>
<tr>
<td>AD</td>
<td>(Host-A, 1)</td>
</tr>
<tr>
<td>DS</td>
<td>(Host-A, 2)</td>
</tr>
<tr>
<td>GY</td>
<td>(Host-D, 2)</td>
</tr>
<tr>
<td>IR</td>
<td>(Host-C, 1)</td>
</tr>
<tr>
<td>KL</td>
<td>(Host-B, 1)</td>
</tr>
<tr>
<td>MA</td>
<td>(Host-B, 2)</td>
</tr>
<tr>
<td>OW</td>
<td>(Host-A, 3)</td>
</tr>
<tr>
<td>TS</td>
<td>(Host-D, 1)</td>
</tr>
<tr>
<td>VP</td>
<td>(Host-C, 2)</td>
</tr>
<tr>
<td>YQ</td>
<td>(Host-B, 3)</td>
</tr>
</tbody>
</table>
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

- **Server**
  - Send \((\text{command, physical-key, data})\);
  - Receive \((\text{status, data})\);
  - ...next Processing

- **Client**
  - Receive \((\text{command, physical-key, data})\)
  - \(\text{Command (physical-key, data)}\)\
    - ...Processing
    - Send \((\text{status, data})\);
Experiment

- To Evaluate B-tree Paging as a Whole
  - Several Costs
    - Communication, Synchronization
      - For Parallel Processing
    - B-tree Paging, Data Migration
      - Fundamental Framework of Parallel Environment

- Executing B-tree Paging in One host
  - Comparing with conventional B-tree
Experimental Environment

- FreeBSD 4.1, Pentium □ (1GHz), IDE Ultra ATA-100 Hard Disk
- B-tree
  - order: 50, data entry:pointer: 4 byte
- 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
  – $10^3 \leq n \leq 10^6$
• 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

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

– Depth of Page Map
  • Same Depth of the Upper sub-tree
    – Same number of sub-tree
Discussion (Querying)

<table>
<thead>
<tr>
<th>Depth of B-tree</th>
<th>Layer of Sub-tree</th>
<th>Depth of Page Map</th>
<th>Expect Ratio</th>
<th>Experimental Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>150%</td>
<td>158%</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>166%</td>
<td>177%</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>2</td>
<td>200%</td>
<td>209%</td>
</tr>
</tbody>
</table>

- 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%

<table>
<thead>
<tr>
<th>Layer of sub-tree</th>
<th>Overhead</th>
<th>Splitting Sub-tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>80%</td>
<td>without Splitting</td>
</tr>
<tr>
<td>2</td>
<td>~130%</td>
<td>with Splitting</td>
</tr>
</tbody>
</table>

- 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