Load Balancing in Indexing for Content-based Video Search on Peer-to-Peer Networks

นายชัยยุทธ ประดิษฐ์ทองงาม
Faculty of Engineering, Chiang Mai University, Thailand
chaiyut@grad.cmu.ac.th
Outline

- Introduction
- Problem
- Solution
- Experimental
- Conclusion
Introduction

- Video Search
Introduction

- Video Search
Introduction
Introduction
Problem

OVERLOADED!
Problem

- Fully replicated
Problem

OVERLOADED!

Index
1  
2  
5  
9  

Index
1  
2  
5  
9  

Index
1  
2  
5  
9  

Index
1  
2  
5  
9  

OVERLOADED!
Solution

- Peer-to-Peer
Solution

- Index construction
  - Extract video frame
  - Extract feature each frame
  - Find cluster centroid by K-Means
  - Find optimal reference point by PCA
  - Map high dimension to one dimension for $B^+$-Tree by distance from cluster centroid to optimal reference point
Solution

- Query
  - Extract video frame (query)
  - Extract feature
  - Find key $x$ by distance from feature to optimal reference point
  - Search between $|x - \varepsilon|$ and $|x + \varepsilon|$
Solution

- BATON support range query, Ex. query 74 - 80

Right Routingtable

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>m</td>
<td>61 - 68</td>
</tr>
<tr>
<td>1</td>
<td>n</td>
<td>75 - 81</td>
</tr>
</tbody>
</table>

27/09/53 Load Balancing in Indexing for Content-based Video Search on Peer-to-Peer Networks
Solution

- Binary balanced tree structure

Definition: tree is balanced if and only if at any node in the tree, the height of its two subtrees differ by at most one.
Solution

- Index Construction
  - Extract video frame
  - Extract feature each frame
  - Find cluster centroid by K-Means
  - Find optimal reference point by PCA
  - **Map high dimension to one dimension for B^+ -Tree by distance from cluster centroid to optimal reference point**
Solution

- $B^+$-Tree
Solution

- B-Tree
BATON Insert Algorithm

1. Let new_node be the joining node.
2. Let host_node be the node which new_node is to be joined to.
3. If isFull(\text{left\_routing\_table}(\text{host\_node})) and isFull(\text{right\_routing\_table}(\text{host\_node})) and
   \(((\text{host\_node}.\text{left\_child} = \text{NULL}) \text{ or } (\text{host\_node}.\text{right\_child} = \text{NULL})) \text{ Then},
4. accept new_node as a child of host_node and split the B-tree structure.
5. Else
6. If (NOT isFull(\text{left\_routing\_table}(\text{host\_node}))) or
   (NOT isFull(\text{right\_routing\_table}(\text{host\_node}))) \text{ Then},
7. forward the join request of new_node to the parent node of host_node.
8. Else
9. let m be a node in left_route_table(\text{host\_node}) or
   right_route_table(\text{host\_node}) not having enough children.
10. If there exist m \text{ Then},
11. forward the join request of m.
Split B-Tree

1. Let \( t \) be the B-tree to be split.
2. Let \( \text{new\_node} \) be the joining node.
3. \textbf{If} the \( \text{new\_node} \) is to be joined as a left child \textbf{Then},
4. \hspace{1cm} \textbf{If} \( t.\text{root}.\text{left\_child} = \text{NULL} \) \textbf{Then},
5. \hspace{2cm} \text{rebuild index } t.
6. \hspace{2.5cm} \textbf{End If}
7. \hspace{1cm} \textbf{If} \text{Integer\_part}(t.\text{root}.\text{no\_entries}/3) = 0 \textbf{Then},
8. \hspace{1.5cm} \text{assign the new B-tree rooted at } t.\text{root}.\text{left\_child} \text{ to } \text{new\_node}.
9. \hspace{2.5cm} \textbf{Else}
10. \hspace{3.5cm} \text{assign the new B-tree rooted at } t.\text{root}.\text{entries}[\text{Integer\_part}(t.\text{root}.\text{no\_entries}/3)] \text{ to } \text{new\_node}.
11. \hspace{1cm} \textbf{End If}
12. \hspace{1cm} \textbf{Else}
13. \hspace{2cm} \textbf{If} \( t.\text{root}.\text{right\_child} = \text{NULL} \) \textbf{Then},
14. \hspace{1.5cm} \text{rebuild index } t.
15. \hspace{2.5cm} \textbf{End If}
Insert Operation

\[
\text{root.no\_entries} = 2 \\
\frac{2}{3} = 0
\]

\[
\text{root.no\_entries} = 2 \\
\frac{2}{2} = 1
\]
BATON Departure Algorithm

1. Let d_node be the departing node.
2. If (d_node.left_child = NULL and d_node.right_child = NULL and
   NOT exists neighbor_node of d_node in left_routing_table(d_node)
   or right_routing_table(d_node) having child) Then,
3.   If NOT exists neighbor_node of d_node in left_routing_table(d_node)
   or right_routing_table(d_node) having child) Then,
4.   transfer the B-tree structure of d_node to be merge with its parent node
   and depart the network.
5. Else
6.   find replacement node from its child nodes.
7.   End If
8. Else
9.   find replacement node from its child nodes.
10. End If
BATON Find and Replacement

1. Let d_node be the departing node.
2. If(d_node.left_child != NULL) Then,
3. forward the replacement request to d_node.left_child.
4. Else If(d_node.right_child != NULL)
5. forward the replacement request to d_node.right_child.
6. Else
7. Let m be a node in left_routing_table(d_node)
   or right_routing_table(d_node) not having enough children.
8. If there exist m Then,
9. forward the replacement request to a child of m.
10. Else
11. replace d_node with the current leaf node.
12. End If
13. End If
Merge B-tree Algorithm

1. Let \( p\_\text{tree} \) be the B-tree of the parent node.
2. Let \( d\_\text{tree} \) be the B-tree of the departing node.
3. Let \( y \) be \( d\_\text{tree}.\text{root} \).
4. Let \( z \) be \( p\_\text{tree}.\text{root} \).
5. Let \( u = y.\text{get}\_\text{first}\_\text{index}\_\text{entry}() \).
6. \textbf{While} (\( y.\text{has}\_\text{next}\_\text{index}\_\text{entry}() \) \( \neq \) \( \text{NULL} \))
7. \hspace{1em} Let \( \text{temp} = u \).
8. \hspace{1em} Let \( u = y.\text{get}\_\text{next}()\_\text{index}\_\text{entry} \).
9. \hspace{1em} \( z.\text{add} (\text{temp}) \).
10. \hspace{1em} \textbf{If} (\( z.\text{subtree}\_\text{size}() > \text{MAX}\_\text{index}\_\text{entry} \)) \textbf{Then},
11. \hspace{2em} split node.
12. \hspace{1em} \textbf{End If}
13. \hspace{1em} \textbf{End While}
14. \textbf{return} \( p\_\text{tree} \).
Departure Operation
Video Search Algorithm

1. Let min be the minimum key of the query key.
2. Let max be the maximum key of the query key.
3. Let node be the current query node.
4. If node.minkey <= min and min <= node.maxkey Then,
5. Return the physical addresses paired with the keys in the current node.
6. If max >= node.maxkey Then,
7. video_query_range( max, node )
8. End If
9. Else
10. If node.minkey <= min and ( min < node.right_adjacent.minkey or
Min < node.right_child.minkey ) Then,
11. Not found the key in the tree use min in the current minkey node.
12. If max >= node.maxkey Then,
13. video_query_range( max, node )
14. End If
15. Else If node.maxkey < min
16. m = The_farthest_node_satisfying_condition( m.minkey <= min )
17. If there exist m Then,
18. Propagate the query to the node m.
19. Else
20. If there exist node.right_child Then,
video query range Algorithm

1. Let $k$ be max key for query.
2. Let node be the current query node.
3. Let $L(node)$ be load of node.
4. Let threshold $= \left\lfloor c \delta^i \right\rfloor$
5. While node.maxkey $> k$
6. Return the physical addresses paired with the keys in the current node.
7. If $L(node)i > \text{threshold}$ Then,
8. ladbalancing(node)
9. End If
10. node = node.right_adjacent
11. End While
Load Balancing Algorithm

1. Let $m$ be a lightly loaded node.
2. departure($m$)
3. forcejoin($m$)
4. If BATON tree imbalance Then,
5. restructuring($m$)
6. End If
Restructure Algorithm

1. Let m be
2. Let n be m.right_adjacent.
3. If isFull(left_routing_table(n)) and isFull(right_routing_table(n)) and n.left_child = NULL Then,
4. Add m to child of n.
5. Else If isFull(left_routing_table(n)) and isFull(right_routing_table(n)) and n.right_child = NULL Then,
6. replace(n, m) and Add to child of m.
7. Else
8. replace(n, m) and call restructuring() again.
9. End If
Load balancing

Restructuring
Experiment

• ทดลองกับวิดีโอจำนวน 1000 วิดีโอ
  • Frame rate 25 fps.
  • Video length 10s, 15s and 30s
Load balancing control factor change

- 100 keys, 0 - 0.1
Query range change

- Delta = 2, 100 keys, 50 peer
Query number change

- Delta = 2, 0 – 0.5, 50 peer
Conclusion

- Using B-tree instead $B^+$-tree
- Framework index videos P2P model
- Our approach more efficient than comparing approach
Thank You

Q&A