Advanced Application
Indexing and Index Tuning
High-dimensional Indexing

• Requirements
  o Fast range/window query search (range query
  o Fast similarity search
    ▪ Similarity range query
    ▪ K-nearest neighbour query (KNN query)
Feature Base Similarity Search

Complex Objects

Feature Vectors

Index for range/similarity Search

Feature extraction and transformation

Index construction

Similarity Queries
Retrieval by Colour

E.g. Function FotoFind in geofoto

Given a sample image

Similarity Search based on sample image in color composition
Window/Range query: Retrieve data points fall within a given range along each dimension.

Designed to support range retrieval, facilitate joins and similarity search (if applicable).
• Similarity queries: Similarity range and KNN queries

- **Similarity range query**: Given a query point, find all data points within a given distance $r$ to the query point.

- **KNN query**: Given a query point, find the K nearest neighbours, in distance to the point.
Cost Factors

- Page accesses
- CPU
  - Computation of similarity
  - Comparisons
Similarity Measures

For dimensions which are independent and of equal importance: $L_p$ metrics

- $L_1$ metric -- Manhattan distance or city block distance
  $D_1 = \sum |x_i - y_i|$

- $L_2$ metric -- Euclidean distance
  $D_2 = \sqrt{\sum (x_i - y_i)^2}$

Histogram: quadratic distance matrix to take into account similarities between similar but not identical colours by using a ‘color similarity matrix’ $D = (X-Y)^T A (X-Y)$
Similarity Measures

For dimensions that are interdependent and varying importance: Mahalanobis metric

\[ D = (X - Y)^T C^{-1} (X - Y) \]

\(C\) is the covariance matrix of feature vectors

If feature dimensions are independent:

\[ D = \sum (x_i - y_i)^2 / c_i \]
Roots of Problems

- Data space is sparsely populated.
- The probability of a point lying within a range query with side $s$ is $\Pr[s] = s^d$
- When $d = 100$, a range query with 0.95 side only selects 0.59% of the data points
Roots of Problems

• Small selectivity range query yields large range side on each dimension.
  e.g. selectivity 0.1% on $d$-dimensional, uniformly distributed, data set $[0,1]$, range side is $\sqrt[20]{0.001}$.

  e.g. $d > 9$, range side > 0.5
       $d = 30$, range side = 0.7943.
The probability of a point lying within a spherical query $\text{Sphere}(q, 0.5)$ is

$$\Pr[s] = \frac{\sqrt{\pi^d} \ (1/2)^d}{(d/2)!}$$

where $d$ is the dimension of the space.
Roots of Problems

- Low distance contrast: as the dimensionality increases, the distance to the nearest neighbour approaches the distance to the farthest neighbor.
- Difference between the nearest data point and the farthest data point reduces greatly (can be reasoned from the distance functions).
Curse of Dimensionality on R-trees

Overlap

Query Performance
The performance of the R-tree based structures degrades with increasing dimensionality.

Sequential scan is much more efficient.
Approaches to High-dimensional Indexing

- Filter-and-Refine Approaches
- Data Partitioning
- Metric-Based Indexing
- Dimensionality Reduction
Approaches to Solving the Problem

- Increase Fan-Out by increasing the node size, as in X-trees. --- still have the same behaviour of R-tree, and performance degrade as the volume of data increases drastically.

- Mapping high-dimensional points into single-dimensional points by using space-filling curves methods or projection.
  - Transformation in high-dimensional space is expensive.
  - The number of sub-queries generated can be very big.

- **Sequential scan** seems like a reasonable --- but, needs to search the whole data file -- affected volume is 100%.
Basic Definition

- **Euclidean distance:**
  \[ \text{dist}(p_1, p_2) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + \cdots + (x_d - y_d)^2} \]

- **Relationship between data points:**
  \[
  \begin{align*}
  \text{dist}(p_1, p_2) &= \text{dist}(p_2, p_1) \\
  \text{dist}(p_1, p_1) &= 0 \\
  0 < \text{dist}(p_1, p_2) &\leq \sqrt{d} \\
  \text{dist}(p_1, p_3) &\leq \text{dist}(p_1, p_2) + \text{dist}(p_2, p_3) \\
  \forall p_1, p_2 \in \text{Points} \\
  \forall p_1 \in \text{Points} \\
  \forall p_1, p_2 \in \text{Points}; p_1 \neq p_2 \\
  \forall p_1, p_2, p_3 \in \text{Points}
  \end{align*}
  \]
Basic Concept of iDistance

- Indexing points based on similarity

\[ y = i \times c + \text{dist} (S_i, p) \]
Selection of Reference Points

Space Partitioning

Data Partitioning
KNN Searching

This figure illustrates how the searching region is enlarged till getting K NNs.
Main Memory Indexing
“...the typical computing engine may have one terabyte of main memory. ‘Hot tables’ and most indexes will be main-memory resident. This will require storage to be rethought. For example, B-tress are not the optimal indexing structure for main memory data.”
### Memory Hierarchy

<table>
<thead>
<tr>
<th>Block Size</th>
<th>L1 Cache</th>
<th>L2 Cache</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16-32 B</td>
<td>32-64 B</td>
<td>4-64 KB</td>
</tr>
<tr>
<td><strong>Size</strong></td>
<td>16-64 KB</td>
<td>256K-8MB</td>
<td>-32GB</td>
</tr>
<tr>
<td><strong>Backing Store</strong></td>
<td>L2</td>
<td>Memory</td>
<td>Disks</td>
</tr>
<tr>
<td><strong>Miss Penalty</strong></td>
<td>5-20 cycles</td>
<td>40-200 cycles</td>
<td>-6M cycles</td>
</tr>
</tbody>
</table>
Address translation

* Translation Lookahead Buffer (TLB) is an essential part of modern processor.
* TLB contains recently used address translations.
* Poor memory reference locality causes more TLB misses.
* Cost of TLB miss is about 100 CPU cycles.
T-tree (1986)

<table>
<thead>
<tr>
<th>mink</th>
<th>data1</th>
<th>……</th>
<th>dataK</th>
<th>maxk</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>bf</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

- parent ptr
- left child ptr
- right child ptr

T-tree has the same tree structure as AVL-tree.
CSB⁺-tree (sigmod’2000)
Other Basic Indexes

- Hash-tables
  - Static table size, with overflow chains

\[ \Theta(1 + \alpha) \]
Other Basic Indexes

*Bitmap Index*
- Bitmap with bit position to indicate the presence of a value
- Counting bits with 1’s
- Bit-wise operations
- More compact than trees-- amenable to the use of compression techniques

*Join indexes*
Index Tuning

- Index issues
  - Indexes may be better or worse than scans
  - Multi-table joins that run on for hours, because the wrong indexes are defined
  - Concurrency control bottlenecks
  - Indexes that are maintained and never used
Information about indexes...

- Application codes
- V$SQLAREA  -- look for the one with high # of executions
- INDEX_STATS: meta information about indexes
- HASH_AREA_SIZE
- HASH_MULTIBLOCK_IO_COUNT
- ...home work
Clustered / Non clustered index

Clustered index (primary index)
- A clustered index on attribute X co-locates records whose X values are near to one another.

Non-clustered index (secondary index)
- A non-clustered index does not constrain table organization.
- There might be several non-clustered indexes per table.
Dense / Sparse Index

Sparse index

- Pointers are associated to pages

Dense index

- Pointers are associated to records
- Non clustered indexes are dense
Index Implementations in some major DBMS

SQL Server
- B+-Tree data structure
- Clustered indexes are sparse
- Indexes maintained as updates/insertions/deletes are performed

DB2
- B+-Tree data structure, spatial extender for R-tree
- Clustered indexes are dense
- Explicit command for index reorganization

Oracle
- B+-tree, hash, bitmap, spatial extender for R-Tree
- Clustered index
- Index organized table (unique/clustered)
- Clusters used when creating tables.

TimesTen (Main-memory DBMS)
- T-tree
Types of Queries

Point Query

SELECT balance
FROM accounts
WHERE number = 1023;

Range Query

SELECT number
FROM accounts
WHERE balance > 10000
and balance <= 20000;

Multipoint Query

SELECT balance
FROM accounts
WHERE branchnum = 100;

Prefix Match Query

SELECT *
FROM employees
WHERE name = ‘J*’;
More Types of Queries

* Extremal Query

```
SELECT *
FROM accounts
WHERE balance =
    max(select balance from accounts)
```

* Ordering Query

```
SELECT *
FROM accounts
ORDER BY balance;
```

* Grouping Query

```
SELECT branchnum, avg(balance)
FROM accounts
GROUP BY branchnum;
```

* Join Query

```
SELECT distinct branch.adresse
FROM accounts, branch
WHERE
    accounts.branchnum =
    branch.number
and accounts.balance > 10000;
```
Index Tuning -- data

Settings:

`employees(ssnum, name, lat, long, hundreds1, hundreds2);
clustered index c on employees(hundreds1)
  with fillfactor = 100;
nonclustered index nc on employees (hundreds2);
index nc3 on employees (ssnum, name, hundreds2);
index nc4 on employees (lat, ssnum, name);
```
1000000 rows ; Cold buffer
```
istem Xeon (550MHz,512Kb), 1Gb RAM, Internal RAID controller from Adaptec
(80Mb), 4x18Gb drives (10000RPM), Windows 2000.
Index Tuning -- operations

Operations:

▲ **Update:**
   update employees set name = 'XXX' where ssnum = ?;

▲ **Insert:**
   insert into employees values
   (1003505,'polo94064',97.48,84.03,4700.55,3987.2);

▲ **Multipoint query:**
   select * from employees where hundreds1= ?;
   select * from employees where hundreds2= ?;

▲ **Covered query:**
   select ssnum, name, lat from employees;

▲ **Range Query:**
   select * from employees where long between ? and ?;

▲ **Point Query:**
   select * from employees where ssnum = ?
Clustered Index

- Multipoint query that returns 100 records out of 1000000.
- Cold buffer
- Clustered index is twice as fast as non-clustered index and orders of magnitude faster than a scan.
Positive Points of Clustering indexes

- If the index is sparse, it has less points -- less I/Os
- Good for multipoint queries
  - eg. Looking up names in telephone dir.
- Good for equijoin. Why?
- Good for range, prefix match, and ordering queries
Index “Face Lifts”

- Index is created with fillfactor = 100.
- Insertions cause page splits and extra I/O for each query.
- Maintenance consists in dropping and recreating the index.
- With maintenance performance is constant while performance degrades significantly if no maintenance is performed.
Index Maintenance

- In Oracle, clustered index are approximated by an index defined on a clustered table.
- No automatic physical reorganization.
- Index defined with pctfree = 0.
- Overflow pages cause performance degradation.
Covering Index - defined

- Select name from employee where department = “marketing”
- Good covering index would be on (department, name)
- Index on (name, department) less useful.
- Index on department alone moderately useful.
Covering Index - impact

- Covering index performs better than clustering index when first attributes of index are in the where clause and last attributes in the select.
- When attributes are not in order then performance is much worse.
Positive/negative points of non-clustering indexes

- Eliminate the need to access the underlying table
  - eg. Index on (A, B, C)
  - Select B, C From R Where A = 5.
- Good if each query retrieves significantly fewer records than there are pages in DB
- May not be good for multipoint queries
Examples:

- Table T has 50-bytes records and attribute A has 20 different values which are uniformly distributed. Page size=4K. Is a nonclustering index on A any good?
- Now the record size is 2Kbytes.
Scan Can Sometimes Win

- IBM DB2 v7.1 on Windows 2000
- Range Query
- If a query retrieves 10% of the records or more, scanning is often better than using a non-clustering non-covering index. Crossover > 10% when records are large or table is fragmented on disk – scan cost increases.
Index on Small Tables

- Small table: 100 records, i.e., a few pages.
- Two concurrent processes perform updates (each process works for 10ms before it commits)
- No index: the table is scanned for each update. No concurrent updates.
- A clustered index allows to take advantage of row locking.
Bitmap vs. Hash vs. B+-Tree

Settings:

```sql
CREATE TABLE employees (ssnum, name, lat, long, hundreds1, hundreds2);
CREATE CLUSTER c_hundreds (hundreds2 NUMBER(8)) PCTFREE 0;
CREATE CLUSTER c_ssnum (ssnum INTEGER) PCTFREE 0 SIZE 60;
CREATE CLUSTER c_hundreds (hundreds2 NUMBER(8)) PCTFREE 0 HASHKEYS 1000 SIZE 600;
CREATE CLUSTER c_ssnum (ssnum INTEGER) PCTFREE 0 HASHKEYS 1000000 SIZE 60;
CREATE BITMAP INDEX b ON employees (hundreds2);
CREATE BITMAP INDEX b2 ON employees (ssnum);
```

- 1000000 rows; Cold buffer
- Dual Xeon (550MHz, 512Kb), 1Gb RAM, Internal RAID controller from Adaptec (80Mb), 4x18Gb drives (10000RPM), Windows 2000.
Multipoint query: B-Tree, Hash Tree, Bitmap

- There is an overflow chain in a hash index
- In a clustered B-Tree index records are on contiguous pages.
- Bitmap is proportional to size of table and non-clustered for record access.
**B-Tree, Hash Tree, Bitmap**

- Hash indexes don’t help when evaluating range queries.
- Hash index outperforms B-tree on point queries.
Summary

- Primary means to reduce search costs (I/O and CPU)
- Properties: robust, concurrent, scalable, efficient
- Most supported indexes: Hash, B^+-trees, bitmap index, and R-trees
- Tuning: Usage, Maintenance, Drop/Rebuild, index locking in buffer...