Spatial Query Processing

Harish Doraiswamy
New York University
Outline

• Spatial data and types of spatial queries
• Spatial Index
• *Interactive* spatio-temporal queries
Spatial Data

• Increase in the availability of Urban data sets
  • Sensors such as GPS, mobile phones
  • social media

• Spatial + Temporal attributes
  • Can have >1 spatial and temporal attributes

• Several hundred millions to billions of records
Spatial Data

• Example: NYC taxi data
  • Pick-up and drop-off locations
  • Pick-up and drop-off time
  • Other attributes
  • ~170 million trips per year

• Queries typically involve both spatial and temporal constrains

• **Fast execution** of spatial (spatio-temporal) queries
Demo
Spatial Queries

• Containment based queries
Spatial Queries

• Spatial joins
Spatial Queries

• Nearest neighbor queries

Find 10 closest subway stations
Spatial Index

All spatial Indexes are inspired by traditional indexes.
1-Dimensional Example

Find points between $x$ and $y$
1-Dimensional Example

Find points with value 5
1-Dimensional Example

Binary Search Tree
1-Dimensional Example

Binary Search Tree
1-Dimensional Example

Binary Search Tree
1-Dimensional Example

Binary Search Tree
KD-Tree

• Extension of a Binary Search Tree
• Supports k-dimensional tree
• Focus on 2D
KD-Tree

• At each level split with respect to one of the dimensions
KD-Tree
KD-Tree

• Polygon containment query
  • Search based on Bounding Box
  • Test with query polygon
KD-Tree

• Advantages
  • Handles skewed data
  • Can be extended to any dimension

• Disadvantages
  • Adding new data need not create balanced tree
Traditional Index Analogy

• Binary Tree
Traditional Index Analogy

- Binary Tree $\rightarrow$ KD-Tree
- Hash Index
1-Dimensional Example

• Create bins using a hash function
1-Dimensional Example

- Create bins using a hash function
- Query
  - Identify bin(s) satisfying query constraint
1-Dimensional Example

- Create bins using a hash function
- Query
  - Identify bin(s) satisfying query constraint
  - Search within the bin
Grid Index

• Extension of hash index to higher dimensions
• Hash function is defined by a grid
• Idea
  • Overlay a grid covering the spatial region
  • Assign objects to different grid cells.
Grid Index
Grid Index
## Grid Index

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
# Grid Index

<table>
<thead>
<tr>
<th>Grid</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>9 11</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3 4</td>
</tr>
<tr>
<td>5</td>
<td>8 10 11</td>
</tr>
<tr>
<td>6</td>
<td>10 11 15</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>12 14</td>
</tr>
<tr>
<td>11</td>
<td>16</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>
Grid Index

- Find Cells Intersected
  - 5, 6, 9, 10
- Test all points in these cells

<table>
<thead>
<tr>
<th>Grid Index</th>
<th>Cells Intersected</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>9, 11</td>
</tr>
<tr>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3, 4</td>
</tr>
<tr>
<td>5</td>
<td>8, 10, 11</td>
</tr>
<tr>
<td>6</td>
<td>10, 11, 15</td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>10</td>
<td>12, 14</td>
</tr>
<tr>
<td>11</td>
<td>16</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>6</td>
</tr>
<tr>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>
Grid Index

- Find Cells Intersected
  - 5,6,9,10
- Test all points in these cells

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>10</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>10</td>
<td>11</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>12</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Grid Index

• Advantages
  • Simple
  • Adding new data
Grid Index

- Disadvantages
  - Grid Size?
  - Data skew
Traditional Index Analogy

- Binary Tree ➡ KD-Tree
- Hash Index
Traditional Index Analogy

• Binary Tree $\rightarrow$ KD-Tree
• Hash Index $\rightarrow$ Grid Index
• Binary Tree $\rightarrow$ Quad Tree
  • Each node divides the space into 4 quadrants
  • Each node has 4 children
Traditional Index Analogy

- Binary Tree ➡ KD-Tree
- Hash Index ➡ Grid Index
- Binary Tree ➡ Quad Tree
- B-Tree
Traditional Index Analogy

- Binary Tree $\rightarrow$ KD-Tree
- Hash Index $\rightarrow$ Grid Index
- Binary Tree $\rightarrow$ Quad Tree
- B-Tree $\rightarrow$ R-Tree
R-Tree

• Balanced search tree
• The two children of a given node
  • minimizes the bounding box
  • bounding boxes of two nodes can intersect
• Designed for storage on disk
  • Each node corresponds to a “page”
• Every node satisfies a minimum fill
R-Tree

• Advantages
  • Supports addition / removal of data

• Disadvantages
  • Bounding boxes can intersect => queries can get expensive
  • Especially true in higher dimensions
Spatial Join

• Order trips based on Neighborhoods
Spatial Join

• Set of trips
  • Points data
  • 2 Years of data - ~340 million trips

• Set of Neighborhoods
  • Polygon data
  • ~200 neighborhoods

• How to join these two data sets?
Spatial Join – Approach 1

• Create Index
  • Trips
• For each neighborhood
  • Query for trips within that neighborhood using the index
• Unnecessary point in polygon tests
Spatial Join – Approach 2

• Create Index
  • Neighborhoods

• For each trip
  • Query for neighborhood using the index
  • Add it to the corresponding neighborhood
Using Map Reduce

Hadoop-GIS: A Spatial Data Warehousing System Over MapReduce
Aji et al. VLDB 2013.

- Spatial Join
- Reduce
  - The required processing
- Build index on polygon to speed up the process
  - Small # of polygons

- Map (Point, Polygons)
  - For given point \( p \) find the neighborhood polygon \( P_i \)
  - Emit \((i, p)\)
Using Map Reduce

*Hadoop-GIS: A Spatial Data Warehousing System Over MapReduce*  
Aji et al. VLDB 2013.

- When both tables being joined has large number of objects
  - Low resolution index to generate “tiles”
  - Using partitioning on tiles to run the Map Reduce jobs
- Map
  - Map the objects to be joined onto a specific tile
- Reduce
  - Build index on the tile records
  - Perform join
STIG: Spatio-Temporal Index using GPUs
Spatio-Temporal Queries

• Example: NYC Taxi Data
  • 5 years data
  • ~ 868 million trips

• Queries
  • Find all trips that occurred between lower Manhattan and the two airports, JFK and LGA, during all Sundays in May 2011.
  • Find all trips that occurred between lower Manhattan and the two airports, JFK and LGA, during all Mondays in May 2011.
  • Find all trips that occurred between Midtown and the two airports, JFK and LGA, during all Sundays in May 2011.
  • Find all trips that occurred between Midtown and the two airports, JFK and LGA, during all Mondays in May 2011.
Existing Database Systems

- Use Indexes
- Need multiple indexes for different attributes
  - A spatial index is only on one spatial attribute
  - Need different indexes for temporal attributes
- Queries with constrains on more than one attribute
  - Single select results in linear processing for applying other constraints
  - Multiple selects requires one or more join operations
Find all trips that occurred between lower Manhattan and the two airports, JFK and LGA, during all Sundays in May 2011.
## Existing Database Systems

<table>
<thead>
<tr>
<th>Query</th>
<th>PostgreSQL</th>
<th>ComDB</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>503.9</td>
<td>20.6</td>
</tr>
<tr>
<td>2</td>
<td>501.9</td>
<td>23.3</td>
</tr>
<tr>
<td>3</td>
<td>437.8</td>
<td>21.6</td>
</tr>
<tr>
<td>4</td>
<td>437.1</td>
<td>32.6</td>
</tr>
</tbody>
</table>

Time in Seconds
Goal

• Design a multi-dimensional spatio-temporal index
• Utilize GPUs to support interactive queries
Design

• KD-Tree Index
KD-Tree Index

• Querying using the index
  • Identify the set of potential leaf blocks that satisfies the constraints specified in the query
  • Compute the result set by searching through the identified leaf blocks
KD-Tree Index

• In-Memory implementation
  • Brute force search of leaf blocks on the GPU
  • Search the identified leaf blocks on the GPU
• Support for multiple GPUs
KD-Tree Index

- Out-of-core implementation
- Search through KD-Tree for leaf blocks using CPU
  - Tree nodes orders of magnitude smaller than data
  - Fits into memory
- Search the identified leaf blocks on the GPU
MongoDB Integration
Available: https://www.github.com/harishd10/mongodb

• MongoDB ver. 2.5.0
  • Interfaces under development to support custom indexes

• KD Tree index created when type is specified as “stig”
  • type: new key type to differentiate from default index creation
  • Spatial attributes specified using default tag “2d”
  • Stored in a separate file

• Query optimizer modified to select KD Tree index automatically based on query constraints

• CUDA
MongoDB Integration

• Example – index creation

    db.trips.ensureIndex({type : "kdtree", pickup time : 1,dropoff time : 1,pickup : "2d", dropoff : "2d" },{ name : "taxi-kdtree" })

• Example – query

    db.trips.find(
        { $or : [ { pickup time : {$gt: start1, $lt: end1 } }, { pickup time : {$gt: start2, $lt: end2 } } 
                , { pickup time : {$gt: start3, $lt: end3 } } , { pickup time : {$gt: start4, $lt: end4 } } ] },
        $pickup : { $geoWithin: {$polygon: [ [mx_1,my_1], ..., [mx_n,my_n], [mx_1,my_1] ]} },
        $dropoff : { $geoWithin: {$polygon: [[jx_1,jy_1], ..., [jx_n,mj_n], [jx_1,jy_1], 
                                           [lx_1,ly_1], ..., [lx_n,ly_n], [lx_1,ly_1] ]} } }
    )
Performance - Data

• Taxi
  • 868 million trips from 5 years
  • Index on pickup location, dropoff location, pickup time and dropoff time

• Twitter data
  • Geo-tagged tweets collected between Oct 2012 - Jan 2014
  • Location, time, tweet, + other attributes
  • 1.12 billion tweets
  • Index on tweet location and time

• Query Time - end to end
  • From the time query is submitted till all results are obtained
Performance – Taxi Data

• Queries
  • Find all trips that occurred between lower Manhattan and the two airports, JFK and LGA, during all Sundays in May 2011.
  • Find all trips that occurred between lower Manhattan and the two airports, JFK and LGA, during all Mondays in May 2011.
  • Find all trips that occurred between Midtown and the two airports, JFK and LGA, during all Sundays in May 2011.
  • Find all trips that occurred between Midtown and the two airports, JFK and LGA, during all Mondays in May 2011.
## Performance – Taxi Data

<table>
<thead>
<tr>
<th>Query</th>
<th>MongoDB</th>
<th>PostgreSQL</th>
<th>ComDB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Time</td>
<td>Speed up</td>
</tr>
<tr>
<td>1</td>
<td>0.075</td>
<td>503.9</td>
<td>6718</td>
</tr>
<tr>
<td>2</td>
<td>0.080</td>
<td>501.9</td>
<td>6273</td>
</tr>
<tr>
<td>3</td>
<td>0.067</td>
<td>437.8</td>
<td>6534</td>
</tr>
<tr>
<td>4</td>
<td>0.070</td>
<td>437.1</td>
<td>6244</td>
</tr>
</tbody>
</table>

Time in Seconds
## Performance – Taxi Data

<table>
<thead>
<tr>
<th>Query</th>
<th><strong>MongoDB</strong></th>
<th><strong>PostgreSQL</strong></th>
<th><strong>ComDB</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Time</td>
<td>Speed up</td>
</tr>
<tr>
<td>1</td>
<td>0.120</td>
<td>288.6</td>
<td>2405</td>
</tr>
<tr>
<td>2</td>
<td>0.124</td>
<td>285.9</td>
<td>2305</td>
</tr>
<tr>
<td>3</td>
<td>0.103</td>
<td>274.5</td>
<td>2665</td>
</tr>
<tr>
<td>4</td>
<td>0.108</td>
<td>274.5</td>
<td>2541</td>
</tr>
</tbody>
</table>

*Time in Seconds*
Performance – Twitter Data

• Queries
  • Select all tweets tweeted from Washington DC in June 2013.
  • Select all tweets tweeted from Boston in June 2013.
  • Select all tweets tweeted from Manhattan in June 2013.
# Performance – Twitter Data

<table>
<thead>
<tr>
<th>Query</th>
<th>MongoDB</th>
<th>PostgreSQL</th>
<th>ComDB</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time</td>
<td>Time</td>
<td>Speed up</td>
</tr>
<tr>
<td>1</td>
<td>0.246</td>
<td>161.2</td>
<td>655</td>
</tr>
<tr>
<td>2</td>
<td>0.288</td>
<td>151.2</td>
<td>525</td>
</tr>
<tr>
<td>3</td>
<td>0.558</td>
<td>286.0</td>
<td>512</td>
</tr>
</tbody>
</table>

Time in Seconds