Skyline query processing

Orestis Gkorgkas

Norwegian University of Science and Technology

August 12, 2015
Outline

Skyline queries

Basic algorithms

Multi-threaded algorithms

Skyline in distributed systems

Skyline on Map-Reduce
Outline

Skyline queries

Basic algorithms

Multi-threaded algorithms

Skyline in distributed systems

Skyline on Map-Reduce
Skyline Queries

Users often have multiple search criteria

- Price
- Location
- Cleanliness
- Stars
- Overall quality
Skyline Queries

- Items are usually ranked according to a single feature
- Users get overwhelmed by the number of all possible options
  - It becomes difficult to find the item that suits their needs
- Usually there is no single best item
- But there are many options that are definitely worse than others
- How can we keep only the good options?
  - Skyline queries
If a point $p$ is no worse at all dimensions than a point $q$ and at least better in one, then $p$ dominates $q$.

The skyline consists of all points that are not dominated by any other point.
Skyline

- If a point $p$ is no worse at all dimensions than a point $q$ and at least better in one, then $p$ dominates $q$
- The skyline consists of all points that are not dominated by any other point
Why skyline queries are important?

- Data exploration
  - They provide an overview of the available data
  - They include good options for any use preference
- Multi-criteria decision analysis
  - Especially useful when client’s processing resources are limited
- Route planning
  - Evaluation of multiple traveling costs
Skyline queries variations

- Constraint skyline queries
  - Constrains are imposed to the range of some of the dimensions
  - e.g. Skyline for the hotels are that are not further than 500m from beach
- Skylines over a subset of dimensions
- Dynamic skylines
  - Each dimension corresponds to a function
  - e.g. Restaurants that are the closest to my current position their closing time is as later as possible from current time
- Skylines over joins
- Skyband queries
- ...
Skyline queries over different settings

- Centralized processing
  - Single-thread algorithms
    - Index-free
    - Index-based
  - Multi-threaded algorithms
  - Skylines on GPU
- Distributed processing
  - Peer-to-peer systems
  - Skyline on Map-Reduce
A naive algorithm

- Pairwise comparison of all points
- Can be implemented as a nested SQL query
- High complexity ($O(n^2)$)

```sql
SELECT * FROM HOTELS outer
WHERE outer NOT EXISTS (
  SELECT * FROM HOTELS inner
  inner.distance <= outer.distance AND
  inner.price <= outer.price AND
  (inner.distance < outer.distance OR
  inner.price < outer.price
  ));
```
Outline

Skyline queries

Basic algorithms

Multi-threaded algorithms

Skyline in distributed systems

Skyline on Map-Reduce
Properties of skylines

- **Transitivity**
  - If $A$ dominates $B$ and $B$ dominates $C$ 
    \[ \Rightarrow A \text{ dominates } C \]
  - \[ \text{Sk}(A_1 \cup A_2) = \text{Sk}(\text{Sk}(A_1) \cup \text{Sk}(A_2)) \]
Block Nested Loops

A
B
C
D
E
F

distance

price

Window

0 1 2 3 4 5 6 7 8 9 10

0 1 2 3 4 5 6 7 8 9 10
Block Nested Loops

A, B1, B, C, D, E, F

Window

<table>
<thead>
<tr>
<th>B1</th>
</tr>
</thead>
</table>
Block Nested Loops

<table>
<thead>
<tr>
<th>Window</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
</tr>
<tr>
<td>C3</td>
</tr>
<tr>
<td>D4</td>
</tr>
</tbody>
</table>

A
B
C
D
E
F
Block Nested Loops

A
B
C
D
E
F

<table>
<thead>
<tr>
<th>Window</th>
</tr>
</thead>
<tbody>
<tr>
<td>B1</td>
</tr>
<tr>
<td>C3</td>
</tr>
<tr>
<td>D4</td>
</tr>
</tbody>
</table>

price vs. distance
Block Nested Loops

![Graph showing the relationship between distance and price with points labeled A to F, and a window table showing B, D5, D4.](image-url)
Block Nested Loops

Window

<table>
<thead>
<tr>
<th>B'</th>
<th>D5'</th>
<th>D4'</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Temp. file

<table>
<thead>
<tr>
<th>E3</th>
</tr>
</thead>
</table>

Distance vs. Price

- A
- B
- C
- D
- E
- F

Price

Distance
Block Nested Loops

```
<table>
<thead>
<tr>
<th>Window</th>
<th>Temp. file</th>
</tr>
</thead>
<tbody>
<tr>
<td>B'</td>
<td>E3</td>
</tr>
<tr>
<td>D5'</td>
<td></td>
</tr>
<tr>
<td>D4</td>
<td>E4</td>
</tr>
</tbody>
</table>
```

A
B
C
D
E
F

A
B
C
D
E
F

```
0 1 2 3 4 5 6 7 8 9 10
```

```
price
```

```
distance
```

```
A
B
C
D
E
F
```

```
distance
```

```
price
```

```
```
Block Nested Loops

Window

<table>
<thead>
<tr>
<th>B'</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Temp. file

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>E3</td>
<td>E4</td>
</tr>
</tbody>
</table>

A
B
C
D
E
F

distance

price

B's
distance

E3
E4

window

Temp. file

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B'</td>
<td>D</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Block Nested Loops

![Graph showing the relationship between distance and price with points labeled A to F and a table for Window and Temp. file showing B', D, E, E3, and E4.](image-url)
Block Nested Loops

```
<table>
<thead>
<tr>
<th></th>
<th>B'</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>E3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Window

- A
- B
- C
- D
- E

Temp. file

- E3
- E4
- A
- C
- F

```python
plt.scatter(distance, price, c='blue', label='Block Nested Loops')
plt.xlabel('distance')
plt.ylabel('price')
plt.legend()
```
Block Nested Loops

![Graph showing the relationship between distance and price with points labeled A, B, C, D, E, and F.]

<table>
<thead>
<tr>
<th>Window</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>E3</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>E4</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C</td>
<td></td>
</tr>
<tr>
<td></td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Temp. file</th>
</tr>
</thead>
<tbody>
<tr>
<td>E3</td>
</tr>
<tr>
<td>E4</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>C</td>
</tr>
<tr>
<td>F</td>
</tr>
</tbody>
</table>
How can we improve the performance?

- Select points with large dominance area
- Put “killer” points first in the BNL window
- Perform a sorted access based on a scoring function
- Skyline points always appear before the points they dominate for any monotonic function
- We do not need to process all objects
How can we improve the performance?

- Select points with **large dominance area**
- Put “killer” points first in the BNL window

![Graph showing points with distance to the beach on the x-axis and price on the y-axis, with dominance areas highlighted.](image-url)
How can we improve the performance?

- Select points with **large dominance area**
  - Put “killer” points first in the BNL window
- Perform a sorted access based on a **scoring function**
  - Skyline points always appear before the points they dominate for any monotonic function
  - We do not need to process all objects
How can we improve the performance?

- Select points with **large dominance area**
  - Put “killer” points first in the BNL window
- Perform a sorted access based on a **scoring function**
  - Skyline points always appear before the points they dominate for any monotonic function
  - We do not need to process all objects
How can we improve the performance?

- Select points with **large dominance area**
  - Put “killer” points first in the BNL window
- Perform a sorted access based on a **scoring function**
  - Skyline points always appear before the points they dominate for any monotonic function
  - We do not need to process all objects
Outline

Skyline queries

Basic algorithms

**Multi-threaded algorithms**

Skyline in distributed systems

Skyline on Map-Reduce
Multi-threaded processing
Multi-threaded processing
Multi-threaded processing

<table>
<thead>
<tr>
<th>Block of points</th>
<th>Skyline</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
</tr>
<tr>
<td>D1</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
</tr>
<tr>
<td>B4</td>
<td></td>
</tr>
<tr>
<td>E5</td>
<td></td>
</tr>
</tbody>
</table>

**Step 1:** Each point is compared independently against the skyline

- If dominated by the skyline it is marked for removal

**Step 2:** Each surviving point is compared against the points with better score

- Some unnecessary comparisons can be made
Multi-threaded processing

Step 1: Each point is compared independently against the skyline
- If dominated by the skyline it is marked for removal

Step 2: Each surviving point is compared against the points with better score
- Some unnecessary comparisons can be made
Multi-threaded processing

Block of points

Skyline

Step 1: Each point is compared independently against the skyline
  ▶ If dominated by the skyline it is marked for removal

Step 2: Each surviving point is compared against the points with better score
  ▶ Some unnecessary comparisons can be made
Multi-threaded processing

Block of points

```
<table>
<thead>
<tr>
<th>E3</th>
<th>B5</th>
</tr>
</thead>
<tbody>
<tr>
<td>D5</td>
<td>A</td>
</tr>
<tr>
<td>D4</td>
<td>F</td>
</tr>
<tr>
<td>C4</td>
<td>B2</td>
</tr>
<tr>
<td>B3</td>
<td></td>
</tr>
</tbody>
</table>
```

Skyline

```
<table>
<thead>
<tr>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>
```

**Step 1:** Each point is compared independently against the skyline
  - If dominated by the skyline it is marked for removal

**Step 2:** Each surviving point is compared against the points with better score
  - Some unnecessary comparisons can be made
Multi-threaded processing

Step 1: Each point is compared independently against the skyline
  - If dominated by the skyline it is marked for removal

Step 2: Each surviving point is compared against the points with better score
  - Some unnecessary comparisons can be made
Multi-threaded processing

Block of points

Skyline

Step 1: Each point is compared independently against the skyline
  ▶ If dominated by the skyline it is marked for removal

Step 2: Each surviving point is compared against the points with better score
  ▶ Some unnecessary comparisons can be made
Multi-threaded processing

Block of points

Skyline

- Step 1: Each point is compared independently against the skyline
  - If dominated by the skyline it is marked for removal
- Step 2: Each surviving point is compared against the points with better score
  - Some unnecessary comparisons can be made
Outline

Skyline queries

Basic algorithms

Multi-threaded algorithms

Skyline in distributed systems

Skyline on Map-Reduce
Properties of a good distributed algorithm

- Low latency - Short response time
- Parallelism
- Minimize network traffic
  - Avoid accessing unnecessary nodes
  - Minimize the transferred data
- Minimize local processing time
Distributed skyline queries

Query Originator

Server makes an execution plan
Distributed skyline queries

Query Originator

Collection of MBRs

Server makes an execution plan

Execution Plan

A

B

C
Distributed skyline queries

Server makes an execution plan

Query Originator
Distributed skyline queries

Server makes an execution plan

Query Originator
Distributed skyline queries

Server makes an execution plan

Query Originator
Distributed skyline queries

Server makes an execution plan

Query Originator
Distributed skyline queries

Query Originator

Server makes an execution plan

Execution Plan

A

B

C
Distributed skyline queries
Outline

Skyline queries

Basic algorithms

Multi-threaded algorithms

Skyline in distributed systems

Skyline on Map-Reduce
Skyline on Map-Reduce

- Every job is split into a Map and a Reduce phase
  - Map($k_1, v_1$) $\rightarrow$ list($k_2, v_2$)
  - Reduce($k_2, list(v_2)$) $\rightarrow$ List($v_3$)
- The mappers do not communicate with each other
Skyline on Map-Reduce
Skyline on Map-Reduce
Skyline on Map-Reduce
Skyline on Map-Reduce

Mapper 1: 000001001
Mapper 2: 010010010
Mapper 3: 100100000
Reducer: 110011001

Local skyline:

Global skyline:
Skyline on Map-Reduce

Mapper 1

Mapper 2

Mapper 3

Reducer

110011001

Local skyline

Local skyline

Local skyline

Global skyline
Skyline on Map-Reduce
Skyline on Map-Reduce

Mapper 1 → Local skyline
Mapper 2 → Local skyline
Mapper 3 → Local skyline
Reducer
Global skyline
Conclusion

- Skyline queries return not dominated points
- Efficiency can be improved in many ways
  - Presorting and filtering
  - Indexes
    - Indexes cannot always be used
    - e.g. Data streams and custom user queries
  - Parallelization
    - Multi-threading
    - Distributed systems
- There are many aspects we did not cover
Thank you

Thank you for your attention!
References


Shiming Zhang, Nikos Mamoulis, and David W. Cheung. Scalable skyline computation using