Efficient Continuous Skyline Computation on Multi-Core Processors Based on Manhattan

IPM – HPC Center

Present by

Mohsen Mahmoudi

MEMOCODE'15

The University of Texas at Austin
September 21-23, 2015

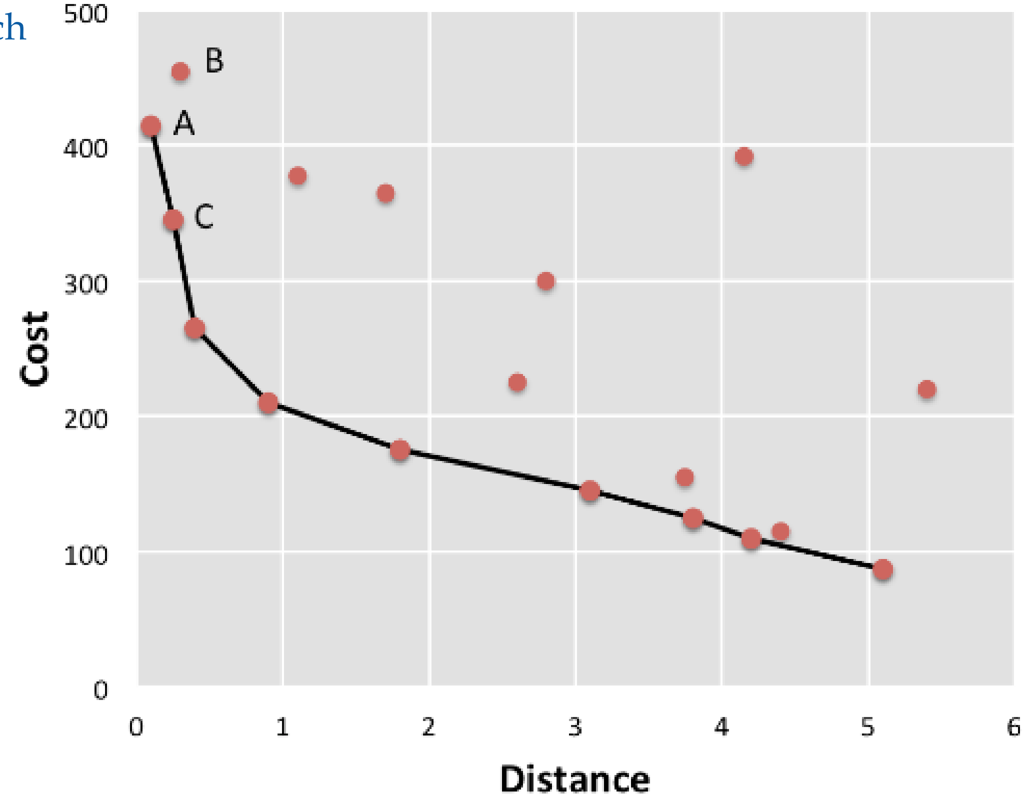
Table of Contents

IPM–HPC Center

Introduction

- **What is Skyline?**

  Find cheap hotels near to the beach



http://www.ece.stonybrook.edu/~pmilder/memocode/

Introduction …

IPM–HPC Center

- **The Skyline Operator**
  - Input :
    Set of points D = $\{d_0, d_1, \ldots, d_{n-1}\}$ with m dimensions
  - Output :
    Subset of D that $\{d_i |\ d_i \in D\ and \nexists\ d^* \in D\ \ s.t.\ d^*\ dominates\ d_i\}$
- **The Continuous Skyline**
  - Each point has arriving time and expiration time
    - The dataset changes over **time**

Proposed Methods

IPM–HPC Center

1. Using "Set" data structure for data points.
   I. does not have data race problem
   II. can be used for sorted data with $O(n \log n)$ complexity

2. Sorting the dataset based on
   I. Added time (arrived time)
   II. Removed time (expiration time)

3. Appointing a pointer to each sorted lists

4. In each step, we proceed on time

Proposed Methods (Cont.)

Algorithm for Skyline Initialization Step

```
Arrival_p = 0, Expiration_p = 0;
FOR (t = start_time TO end_time) DO
    While ( Arrival_time [Arrival[Arrival_p]] <= t){
        Arrival_nodes_list.add(Arrival[Arrival_p]);
        Arrival_p + + ; }
    While (Expiration_time [Expiration[Expiration_p]] <= t) {
        Expiration_node_list.add(Expiration[Expiration_p])
        Expiration_p + +; }
    Update_Skyline(Arrival_nodesss_list, Expiration_node_list);
END DO;
RETURN;
```

Updating Skyline Algorithm

- This problem has a dynamic dataset
- Two phases: Insert and Remove.
- Using Manhattan distance in Insertion and Remove

Proposed Methods

▪ Updating Skyline Elements

✓**Insert process :**

A new entry (p) is checked just with Skyline elements

✓**Remove process :**

in this process two different cases may occur:

- Remove an Skyline element ✗
- Remove a non Skyline element ✓

Proposed Methods …

IPM–HPC Center

- **Manhattan distance**
  - **Base on definition for "dominate" condition :**

$$A \ dominate \ B \overset{if}{\Rightarrow} \sum_{0}^{m-1} A[i] < \sum_{0}^{m-1} B[i]$$

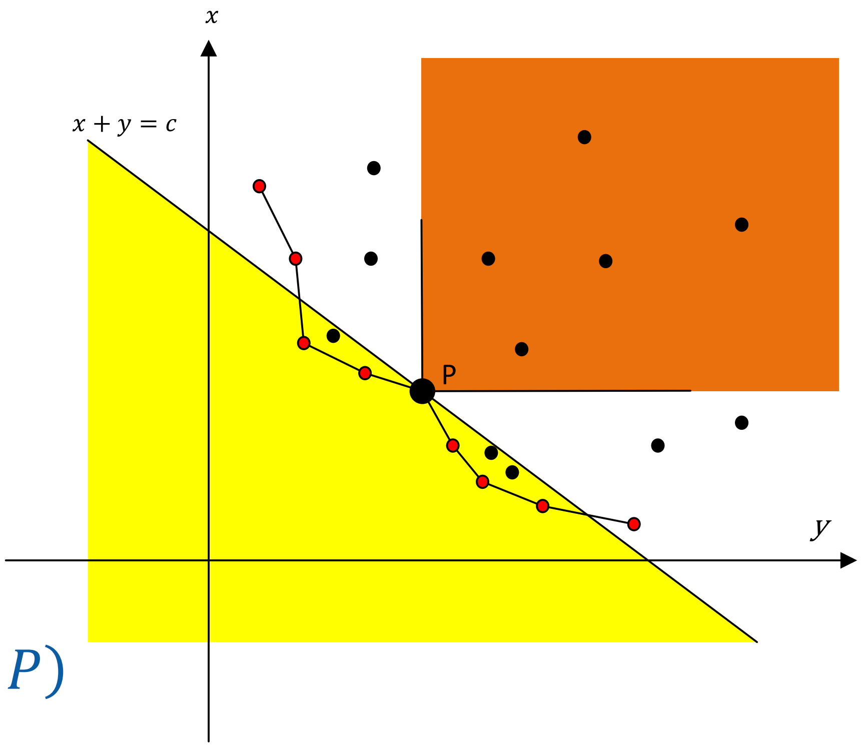
and obviously:

$$if \ \sum_{0}^{m-1} A[i] \geq \sum_{0}^{m-1} B[i] \overset{then}{\Longrightarrow} A \ does \ not \ dominate \ B$$

$$\text{candidate} = \{x \in D \mid P \text{ dominates } x\}$$

$$newS = S \cup \{x \in candidate \mid \nexists x \in S \ (x \text{ dominates } P)\}$$

Using Manhattan distance for pruning points.



Proposed Methods …

- **Parallel Implementation Details**

  – Parallelized the problem over the time.
    - partition the time steps based on number of available cores.

  – We provide two different Parallel solutions

    I. **Static**: fixed overlap

    II. **Dynamic**: set overlap value based on dataset elements.

Implementation Platforms

IPM–HPC Center

We run our implementation on following platforms:

Platform	Cores	Frequency (Ghz)
Intel Corei5-2410	2	2.3
Intel Corei7-960	4	3.20
Intel Core i7-3540M	2	3.0
Intel Xeon X5650	**6**	**2.66**
Intel Xeon E5-2650	8	2.0
AMD Opteron 6386 SE	**16**	**2.8**

Experimental Results

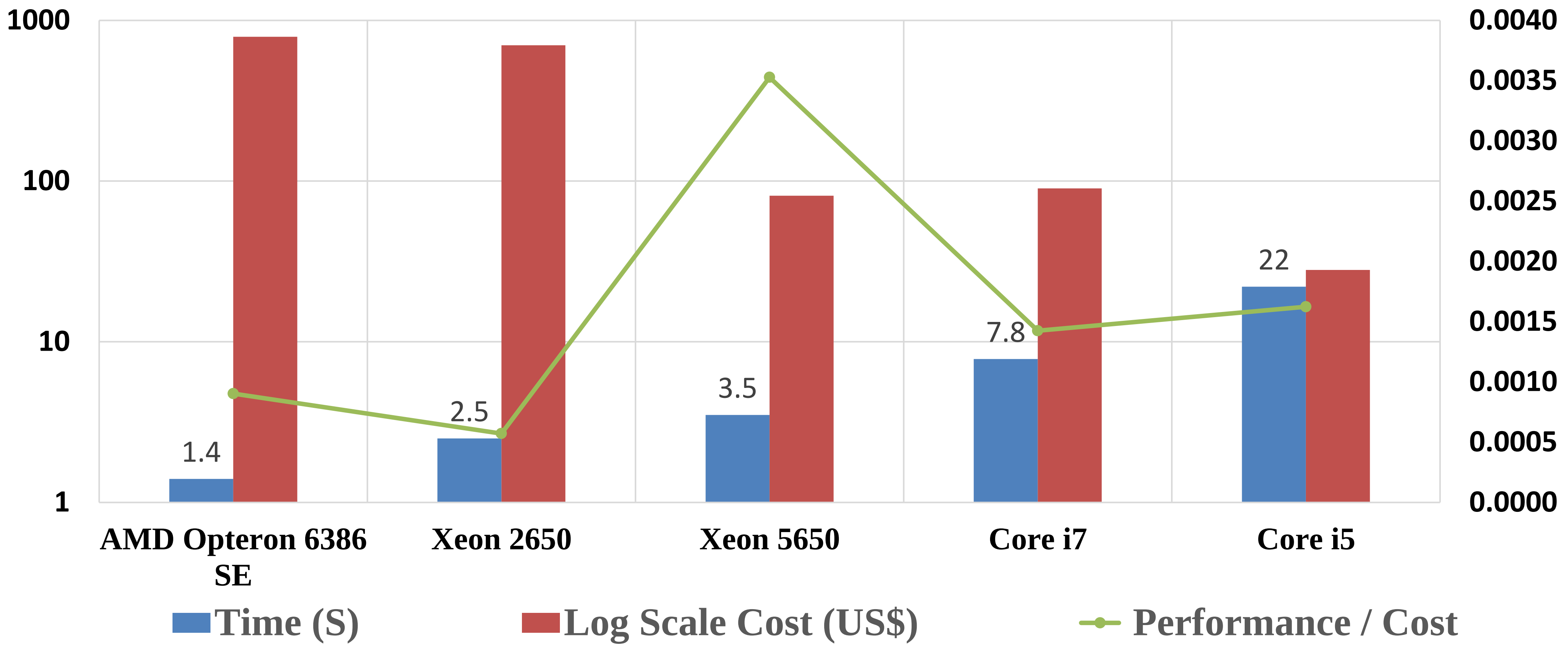
IPM–HPC Center

Reported results for large dataset (800k points)

Design	Platform	Time (Sec) Dynamic	Time (Sec) Static
Naive	Intel Corei7-960	**604800**	**604800**
Our Solution	Intel Corei5-2410M	**23.1**	**22.0**
Our Solution	Intel Corei7-3540M	**16**	**15**
Our Solution	Intel Corei7-960	**8.6**	**7.8**
Our Solution	**Intel Xeon X5650**	**3.9**	**3.5**
Our Solution	Intel Xeon E5-2650	**3.1**	**2.5**
Our Solution	**AMD Opteron 6386 SE**	**1.9**	**1.4**

Experimental Results …

IPM–HPC Center



Conclusion

- Based on provided results:
  - Best Pure-Performance is "AMD Opteron 6386" platform with 432KX speed up.
  - Cost Adjusted Performance is Xeon 5650 platform with 283 Runtime × Cost.

Thanks for your attention!

IPM–HPC Center