Factor Windows: Cost-based Query Rewriting for Optimizing Correlated Window Aggregates

Wentao Wu (Microsoft Research)

Philip A. Bernstein (Microsoft Research)

Alex Raizman (Microsoft)

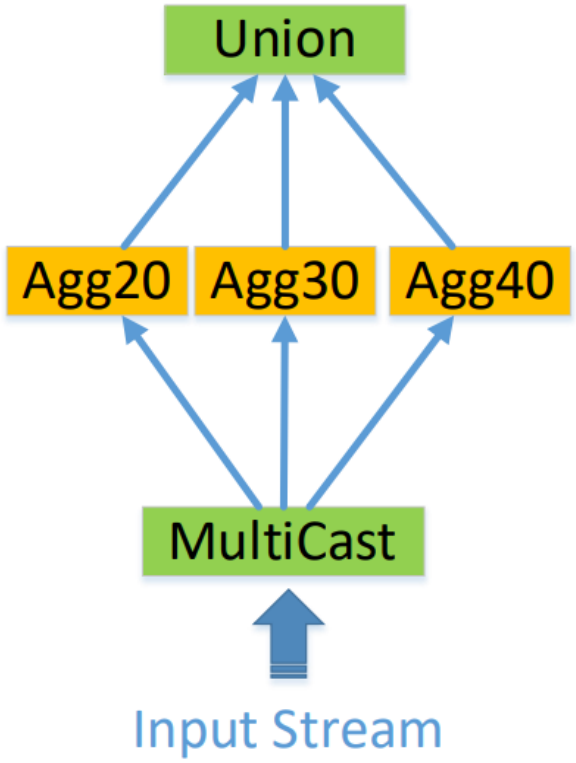
Christina Pavlopoulou (University of California, Riverside)

Scenario: Correlated Window Aggregates

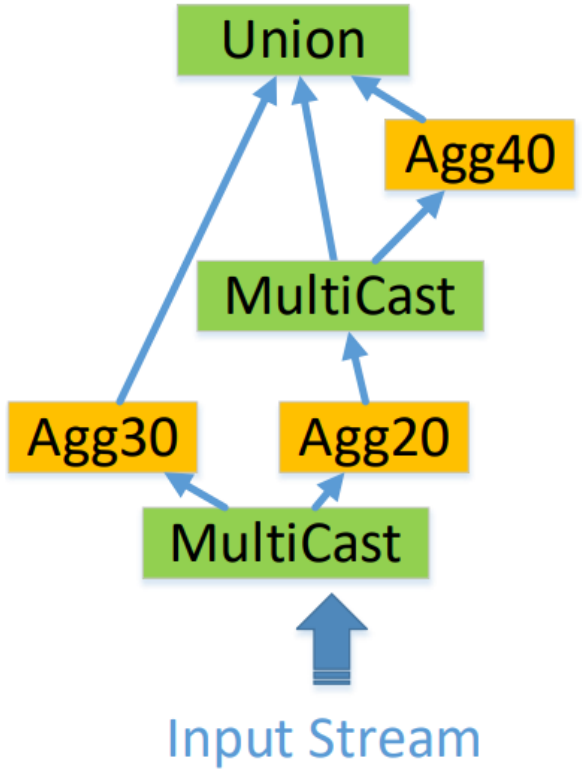
An example “window set” query from Azure Stream Analytics:
https://azure.microsoft.com/en-us/services/stream-analytics/

```
SELECT DeviceID, System.Window().Id, Min(T) AS MinTemp,
FROM Input TIMESTAMP BY EntryTime
GROUP BY DeviceID, Windows(
        Window('20 min', TumblingWindow(minute, 20),
        Window('30 min', TumblingWindow(minute, 30),
        Window('40 min', TumblingWindow(minute, 40))
```

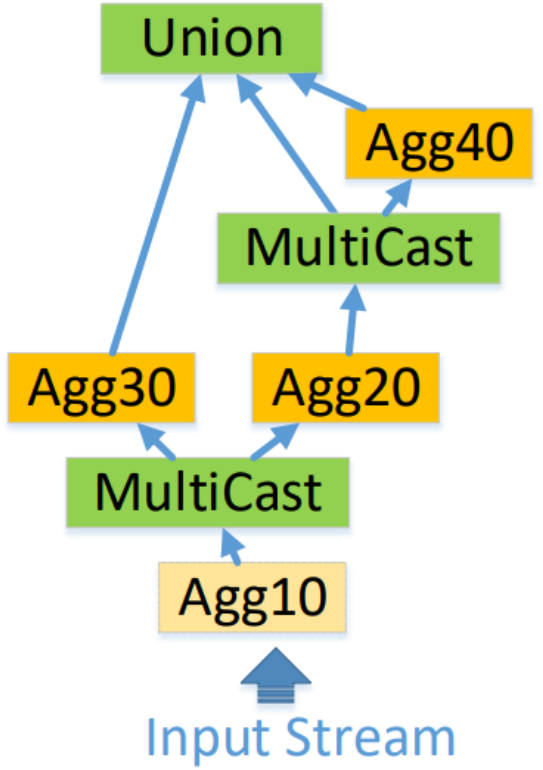
Overview of Our Solution



(a) Default query plan

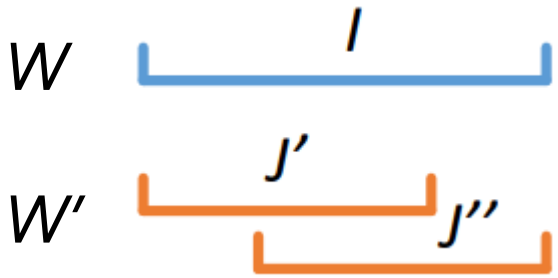


(b) Optimized query plan

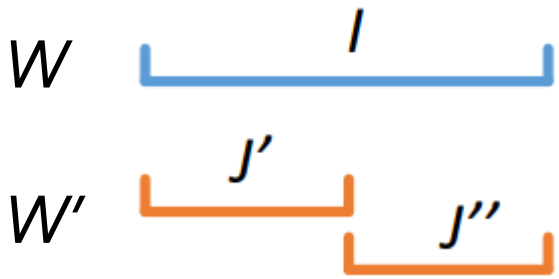


(c) Optimized query plan with factor windows

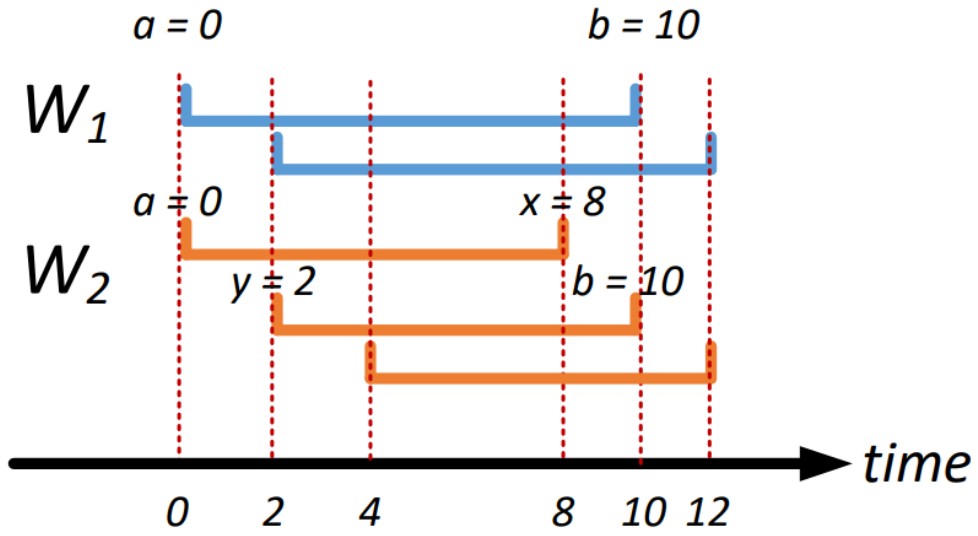
Window Coverage



(a) Window coverage



(b) Window partitioning



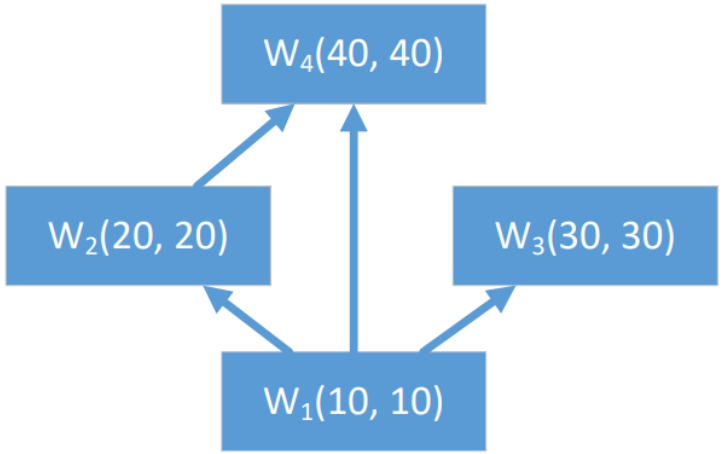
W_1 is "covered by" W_2

Window Coverage Graph (WCG)

- We represent a window as $W\langle r, s\rangle$, where r and s represent W's "range" and "slide".
  - W is a "tumbling" window if $r = s$.
  - W is a "hopping" window if $r > s$.

(a) Tumbling Window (b) Hopping Window

- Window coverage graph (WCG)
  - Vertices represent the windows in the window set.
  - There is an edge from W_2 to W_1 if W_1 is covered by W_2.
  - Example: Consider a window set with four tumbling windows $W_1\langle 10, 10\rangle, W_2\langle 20, 20\rangle, W_3\langle 30, 30\rangle, W_4\langle 40, 40\rangle$.
  - WCG is a directed acyclic graph (DAG).



The WCG representation of the window set

Cost Modeling

- Assume that the cost of computing an aggregate function f is proportional to the number of events processed.

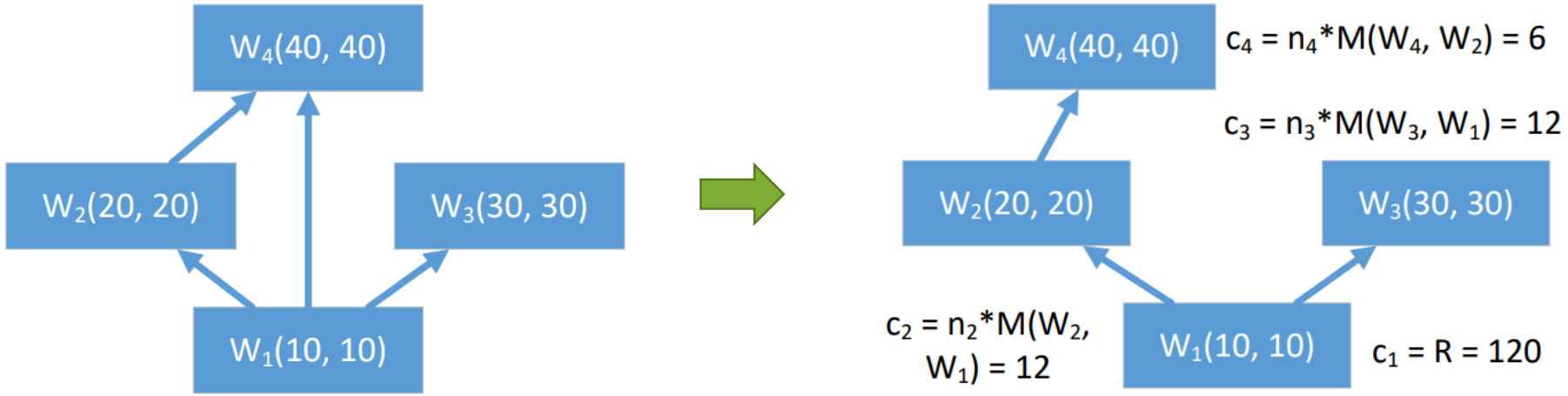
- The "**instance/interval cost**" of window $W\langle r, s\rangle$ is $\eta \cdot r$, where η is the input event rate.

- If W_1 is covered by W_2, the instance cost of W_1 is reduced from $\eta \cdot r_1$ to $M(W_1, W_2) = 1 + \frac{r_1 - r_2}{s_2}$.
  - Intuitively, $M(W_1, W_2)$ represents the number of intervals from W_2 that cover an interval from W_1.

- If W_1 is covered by multiple W_2's, the instance cost of W_1 is reduced to $\min_{W_2} \{M(W_1, W_2)\}$.

Min-Cost WCG

- Example: Consider a window set with $W_1\langle 10, 10\rangle, W_2\langle 20, 20\rangle, W_3\langle 30, 30\rangle, W_4\langle 40, 40\rangle,$.



(a) Initial WCG, cost = 480

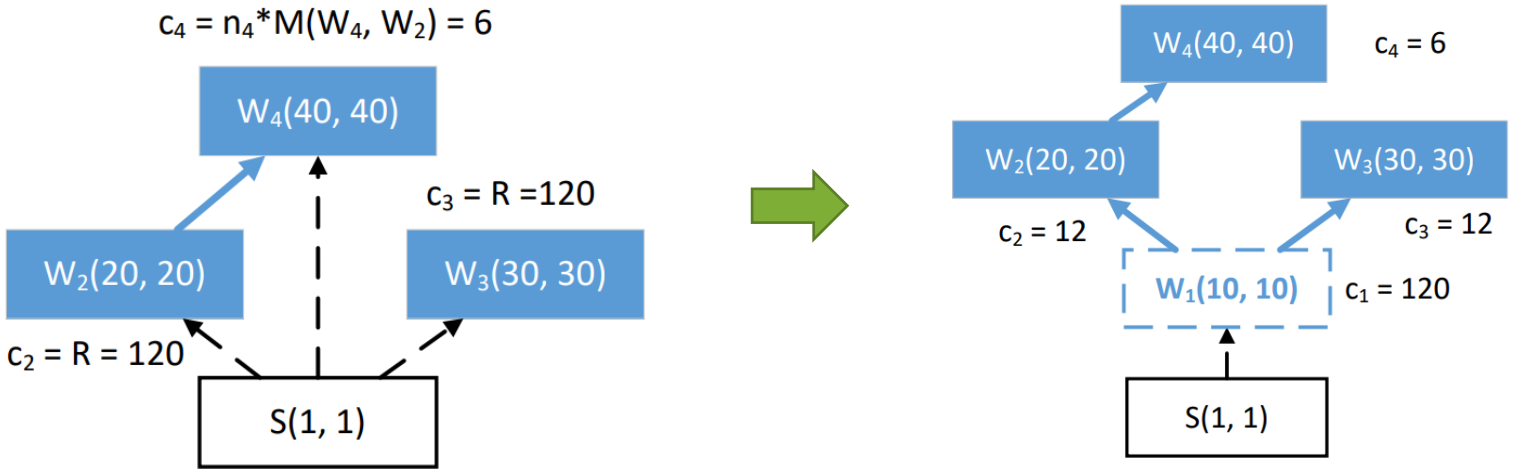
(b) Min-cost WCG, cost = 150

Factor Windows

- Factor window - a window not in the input window set that can help reduce the evaluation cost.

  I.e., it's a window that covers some window(s) in the window set.

- Example: Consider a window set with $W_2\langle 20, 20\rangle, W_3\langle 30, 30\rangle, W_4\langle 40, 40\rangle$.



(a) Initial WCG (i.e., Min-cost WCG w/o factor window), Cost = 246

(b) Min-cost WCG with factor window $W_1\langle 10, 10\rangle$, Cost = 150

Technical Problem: How to Find the Optimal Factor Windows?

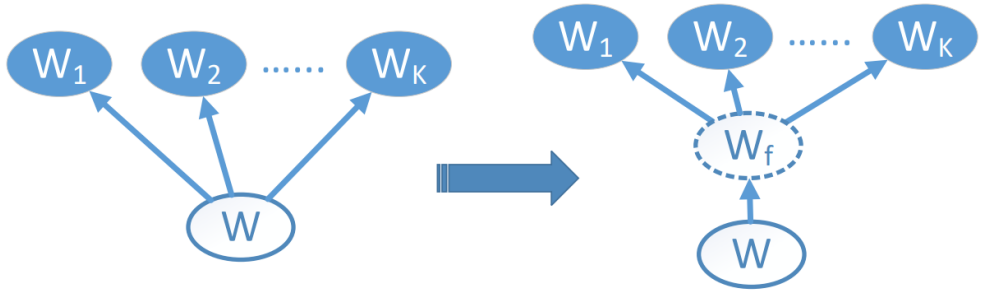
- NP-hard problem (Steiner tree)

- Our "greedy" solution: Find factor windows that are "**locally**" optimal.

  (1) Candidate generation

  (2) Candidate selection

- Special treatment for "**partitioned by**" semantics: See the paper for details.



Candidate Generation

- Find eligible slides S_f.

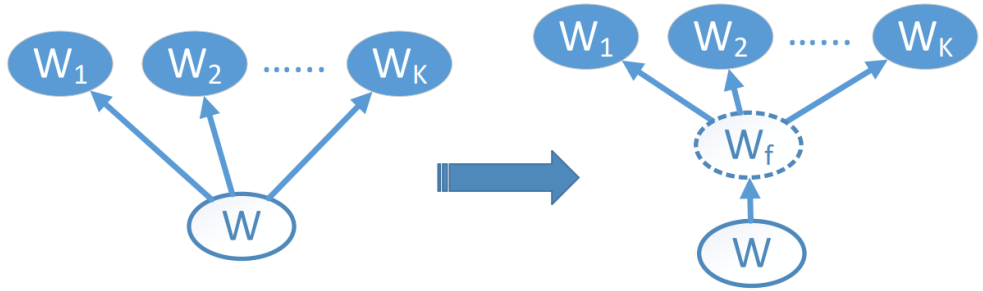
  (1) $s_d = \gcd\{s_1, \ldots, s_K\}$

  (2) $S_f = \{s_f : s_d \bmod s_f = 0 \text{ and } s_f \bmod s_W = 0\}$

- Find eligible ranges R_f for each $s_f \in S_f$.

  (1) $r_{min} = \min\{r_1, \ldots, r_K\}$

  (2) $R_f = \{r_f : r_f \bmod s_f = 0 \text{ and } r_f \leq r_{min}\}$

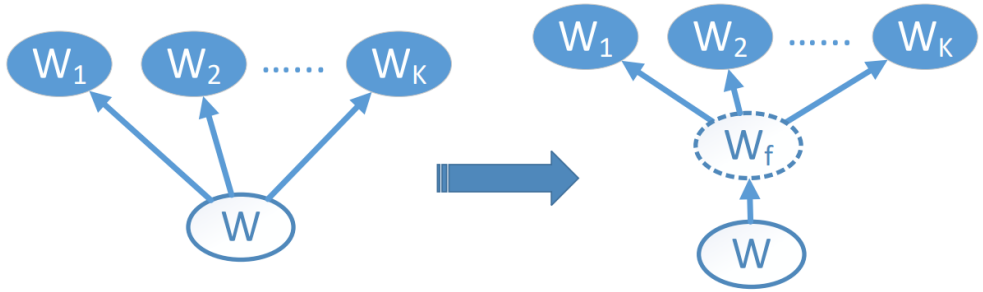


- Construct a candidate factor window $W_f = \langle r_f, s_f \rangle$ for each pair $(r_f \in R_f, s_f)$.

Candidate Selection

- Compute the "**benefit**" δ_f of each candidate factor window W_f.

$$\delta_f = \sum_{j=1}^{K} n_j \cdot \left(\frac{r_j - r_W}{s_W} - \frac{r_j - r_f}{s_f}\right) - n_f \cdot \left(1 + \frac{r_f - r_W}{s_W}\right)$$



- Select the factor window with the maximum benefit.

Put Things Together

1 // Construct the set $\mathcal{W}_f$ of candidate factor windows.
2 $\mathcal{W}_f \leftarrow \emptyset$;
3 $s_d \leftarrow \gcd\{s_1, ..., s_K\}$;
4 $\mathcal{S}_f \leftarrow \{s_f : s_d \bmod s_f = 0 \text{ and } s_f \bmod s_W = 0\}$;
5 $r_{\min} \leftarrow \min\{r_1, ..., r_K\}$;
6 **foreach** $s_f \in \mathcal{S}_f$ **do**
7 $\quad \mathcal{R}_f \leftarrow \{r_f : r_f \bmod s_f = 0 \text{ and } r_f \leq r_{\min}\}$;
8 $\quad$ **foreach** $r_f \in \mathcal{R}_f$ **do**
9 $\quad\quad$ Construct a candidate factor window $W_f\langle r_f, s_f\rangle$;
10 $\quad\quad$ **if** $W_f \leq W$ *and* $W_j \leq W_f$ *for* $1 \leq j \leq K$ **then**
11 $\quad\quad\quad \mathcal{W}_f \leftarrow \mathcal{W}_f \cup \{W_f\}$;
12 // Find the best factor window from $\mathcal{W}_f$.
13 $\delta_f^{\max} \leftarrow 0$, $W_f^{\max} \leftarrow$ `null`;
14 **foreach** $W_f \in \mathcal{W}_f$ **do**
15 $\quad$ Compute the benefit δ_f of W_f using Equation 2;
16 $\quad$ **if** $\delta_f \geq 0$ *and* $\delta_f > \delta_f^{\max}$ **then**
17 $\quad\quad \delta_f^{\max} \leftarrow \delta_f$, $W_f^{\max} \leftarrow W_f$;
18 **return** $W_f^{\max}$;

Candidate Generation

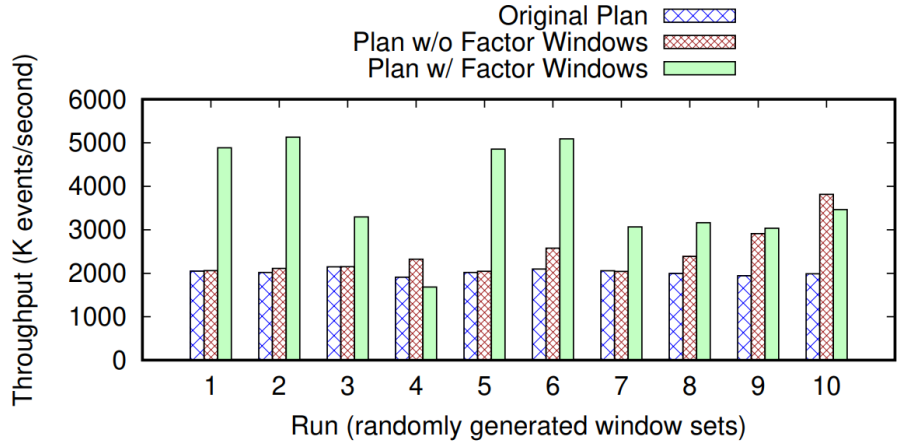
Candidate Selection

Experimental Settings

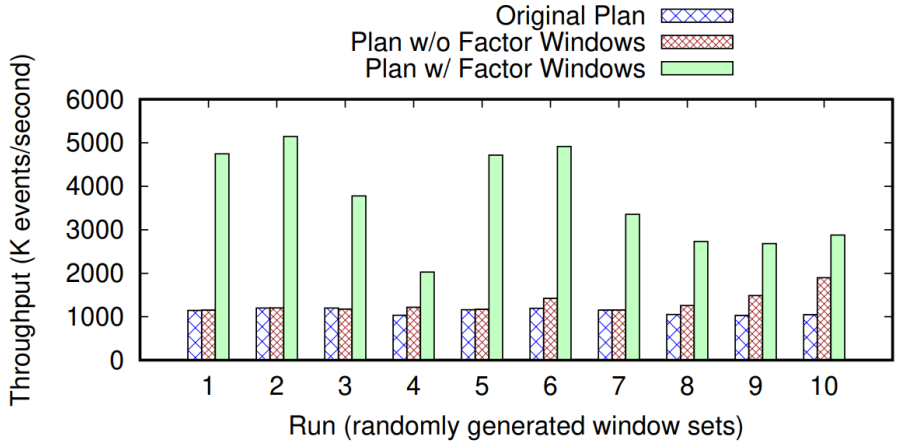
- Experimental Setup
  - *(1) Prototype*: C# implementation that produces optimized query plans (with and without factor windows) represented by Trill expressions.
  - *(2) Performance metric*: We measure the throughput of the original plan and the optimized query plans.
  - *(3) Runtime configuration*: Single-core execution with 2.2 GHz Intel CPUs and 128 GB main memory.
- Datasets
  - *(1) Synthetic Datasets*: Synthetic-1M (1 million events), Synthetic-10M (10 million events).
  - *(2) Real Dataset*: Real-32M (from DEBS 2012 Grand Challenge, with 32 million events).
- Window Sets
  - *(1) RandomGen*: Windows with randomly generated slides and ranges.
  - *(2) SequentialGen*: Windows whose ranges follow a "sequential" pattern (e.g., 10s, 20s, 30s, 40s, …).

Throughput with and without Factor Windows

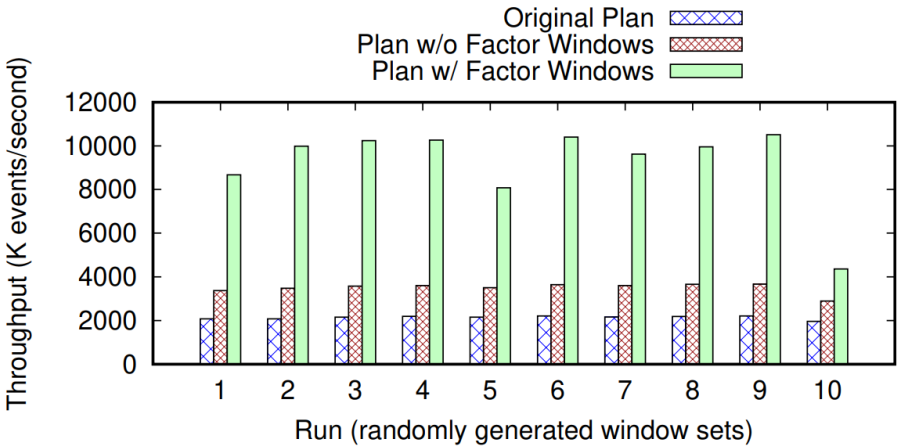
Synthetic10M, Window set size = 5



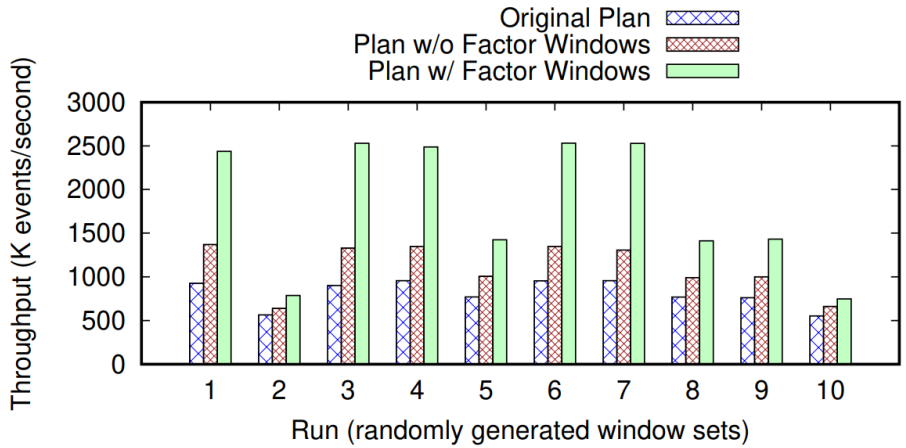
(a) **RandomGen**, "partitioned by"



(b) **RandomGen**, "covered by"



(c) **SequentialGen**, "partitioned by"



(d) **SequentialGen**, "covered by"

Summary of Throughput Results

Synthetic10M
Window set size
= {5, 10}

Setup	w/o FW (Mean)	w/o FW (Max)	w/ FW (Mean)	w/ FW (Max)
R-5-tumbling	1.21×	1.92×	1.85×	2.54×
R-10-tumbling	1.34×	1.77×	1.88×	3.38×
R-5-hopping	1.18×	1.82×	3.26×	4.29×
R-10-hopping	1.34×	1.71×	3.20×	6.15×
S-5-tumbling	1.63×	1.67×	4.28×	4.81×
S-10-tumbling	1.98×	2.05×	7.91×	**9.38**×
S-5-hopping	1.34×	1.48×	2.17×	2.81×
S-10-hopping	1.58×	1.73×	2.92×	3.79×

Real32M
Window set size
= {5, 10}

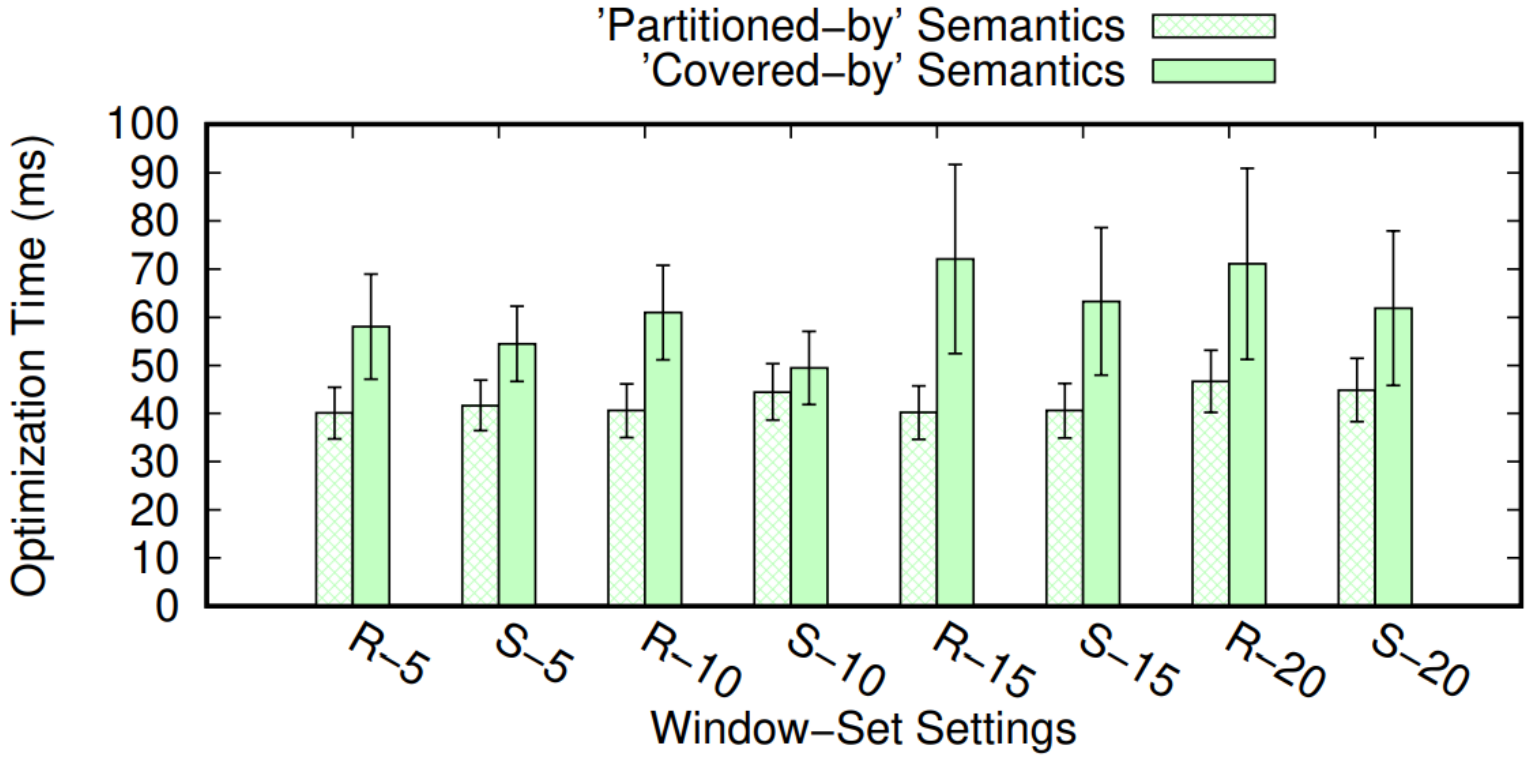
Setup	w/o FW (Mean)	w/o FW (Max)	w/ FW (Mean)	w/ FW (Max)
R-5-tumbling	1.19×	1.78×	1.43×	1.91×
R-10-tumbling	1.30×	1.71×	1.53×	2.86×
R-5-hopping	1.09×	1.39×	1.54×	2.63×
R-10-hopping	1.18×	1.39×	1.46×	3.53×
S-5-tumbling	1.63×	1.67×	4.12×	4.85×
S-10-tumbling	1.90×	1.97×	7.53×	**9.14**×
S-5-hopping	1.12×	1.30×	1.22×	1.77×
S-10-hopping	1.22×	1.51×	1.45×	2.31×

Scalability w.r.t. Window Set Size

Synthetic10M
Window set size
= {15, 20}

Setup	w/o FW (Mean)	w/o FW (Max)	w/ FW (Mean)	w/ FW (Max)
R-15-tumbling	1.55×	1.96×	2.97×	4.34×
R-20-tumbling	1.49×	2.29×	2.10×	4.83×
R-15-hopping	1.55×	1.95×	4.67×	6.59×
R-20-hopping	1.68×	2.20×	4.23×	7.65×
S-15-tumbling	2.43×	2.49×	11.29×	13.83×
S-20-tumbling	2.42×	2.53×	14.28×	**16.82**×
S-15-hopping	1.85×	2.09×	3.51×	4.68×
S-20-hopping	1.91×	2.15×	4.02×	5.32×

Query Optimization Overhead



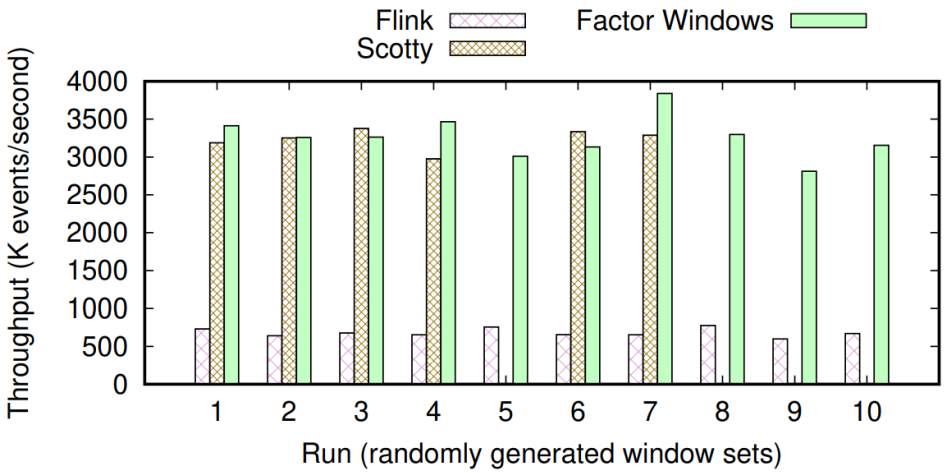
Vary window set size from 5 to 20.

Comparison with Window Slicing

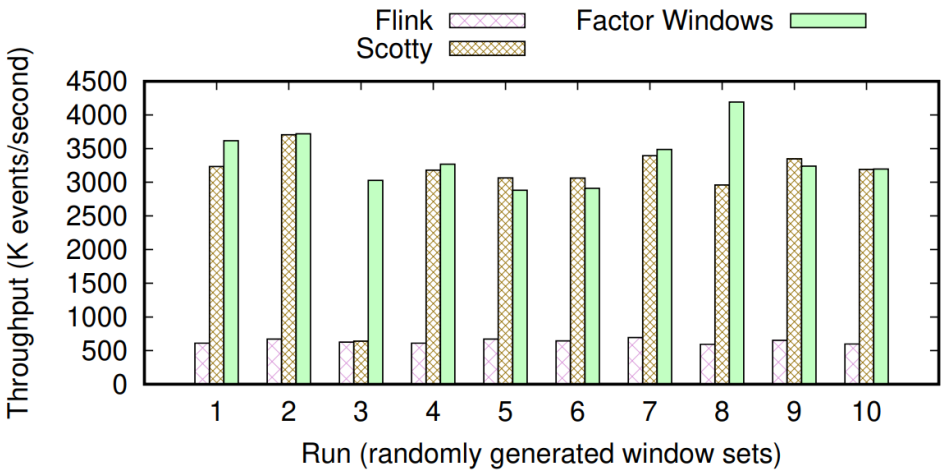
- Window slicing
  - Chop the window into smaller chunks (i.e., slices).
  - Compute the aggregate over the window by aggregating sub-aggregates over the slices.

- Scotty (TODS'21): "General stream slicing"
  - We compare our factor-window based optimization against Scotty on top of Apache Flink.
  - We use the same data generator developed by Scotty for benchmarking its own performance: https://github.com/TU-Berlin-DIMA/scotty-window-processor.

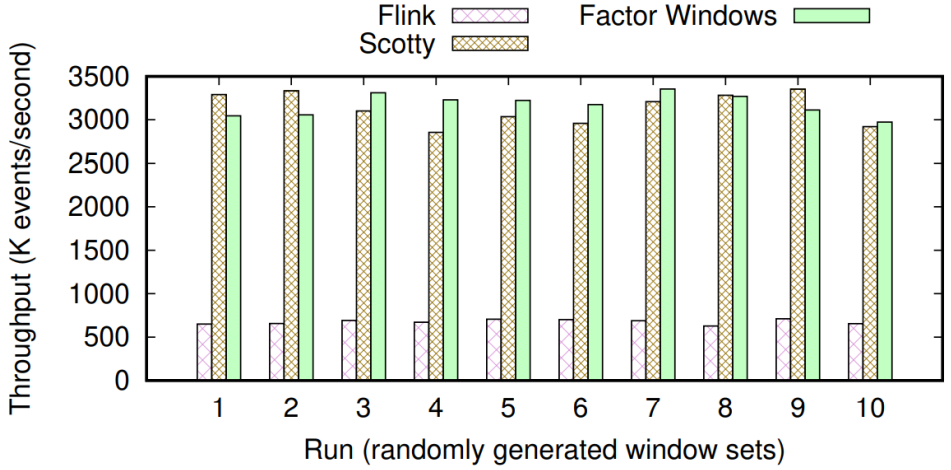
Comparison Results



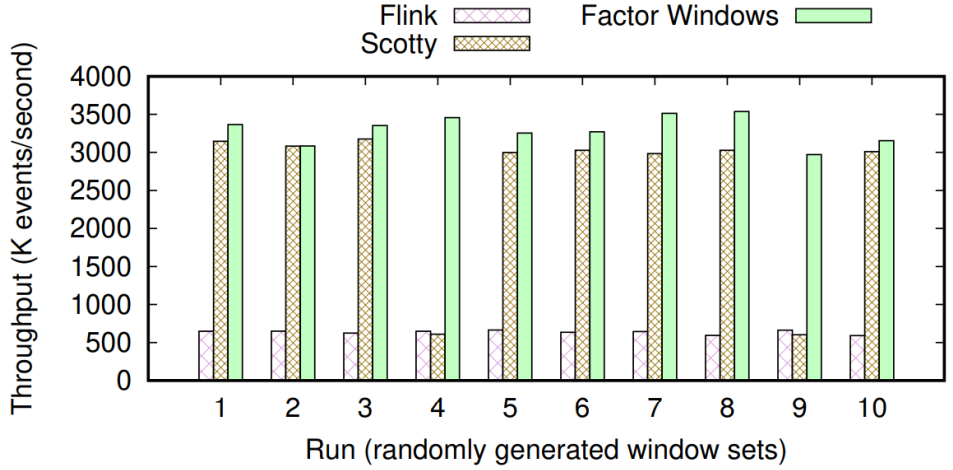
(a) **RandomGen**, "partitioned by"



(b) **RandomGen**, "covered by"



(c) **SequentialGen**, "partitioned by"



(d) **SequentialGen**, "covered by"

Window set size = 10

Conclusion: Summary of Contributions

- We proposed the "window coverage graph" (WCG) abstraction that captures the overlapping relationships between correlated windows.
- We proposed a cost-based optimization framework on top of the WCG abstraction, to minimize the computation cost of multi-window aggregate queries.
- We extended the cost-based framework by considering "factor windows" and developed technologies to find beneficial factor windows.
- We evaluated our techniques on both synthetic and real datasets and demonstrated that the **throughput** of optimized plans can outperform the original plans by up to 16.8x.

Thank you for viewing our presentation! We'd be happy to hear your comments and questions: wentao.wu@microsoft.com.