Hammer Slide: Work- and CPU-efficient Streaming Window Aggregation

Georgios Theodorakis, Alexandros Koliouisis, Peter Pietzuch, Holger Pirk

Large-Scale Data & Systems (LSDS) Group
Imperial College London

http://lsds.doc.ic.ac.uk
<grt17@imperial.ac.uk>

[ADMS 2018 - Monday, August 27, 2018]
It’s a Streaming Problem!

Data-Intensive Applications
- Real-time analytics applications
- Personalised web services
- Network Monitoring
- Fraud Detection

High-throughput & Result Freshness Matter!

Stream Processing
Processing unbounded data

Windows: slice infinite data streams to finite subsets

Tumbling Window

Sliding Window

Every element contributes to one Window Result

Overlapping Window Results
Window Aggregation: a key operator in Streaming

Window Aggregation: accumulate the most recent data

**Naive Window Computation:**
- Recompute from scratch
- Easy to parallelise

**Incremental Window Computation:**
- Intermediate result sharing
- Data and control dependencies

---

**Subtract-on-Evict (SoE):**
$O(1)$ for invertible functions (sum, avg, cnt)

What happens for non-invertible functions (min/max)?
Non-invertible functions? Two-Stacks

Queue maintained as Two-Stacks:
1. **Push** values in **Front Stack**: $O(1)$
2. When Front Stack is full: **Flip Phase** in $O(n)$

```sql
select timestamp, MIN(cpu)
from TaskEvents
```

Input Values  |  Aggregate Values
--- | ---
5 | 1
8 | 1
2 | 1
1 | 1

Front Stack  |  Back Stack
--- | ---
5 | 8
9 | 7
0 | 5
2 | 7
4 | 6
3 | 9
7 | 0

2 7 4 6 3 9 7 0
Two-Stacks: Flip Phase

Queue maintained as Two-Stacks:
1. **Push** values in **Front Stack**: $O(1)$
2. When Front Stack is full: **Flip Phase in** $O(n)$

```
select timestamp, MIN(cpu)
from TaskEvents  
```

[FLIP PHASE] $MIN = 1$
Queue maintained as Two-Stacks:

1. **Push** values in **Front Stack**: $O(1)$
2. When Front Stack is full: **Flip Phase** in $O(n)$
3. **Pop** values from the **Back Stack**: $O(1)$

Flip happens occasionally:
- $O(1)$ amortised time for all operations
- **High throughput for slide 1!**
Problem Statement

What happens when slide > 1?

Window Size 1024

Throughput (10^6 tuples/sec)

Non-Incr
SoE
Two-Stacks

AVG (slide 1)  MIN (slide1)
Problem Statement

Can we have a single unified operator:
- that supports both functions?
- has good performance for varying slides?

What happens when slide > 1?
Do we maintain & use a different version each time?

Can we have a single unified operator:
- that supports both functions?
- has good performance for varying slides?
Microarchitecture analysis
Window Aggregation: Where does time go?

MIN, Window Size 1024, Slide 1

Different patterns than DBs
- Data dependencies
- Control-hazards

Can we reduce the retiring ratio & compute more data at the same time?

% slots retire:
- many operations commit per cycle
How do we improve CPU-efficiency?

- **Vectorise** the computation within a slide (>1)
  - SIMD instructions
- **Reduce** L1 memory stalls
How to represent state?

Not suitable for SIMD! ➔ Isolate data model from physical representation!

- Enable optimisations: SIMD instructions
- Process state that grows arbitrarily with pointer manipulation
- CPU-cache-friendly data layout
- Remove unnecessary writes from flip phase

Comparison with STL Stacks: 86% improvement
Reduce footprint and operations in L1 cache!

1. Store window state in a **columnar format**
   - a) easy to compress
   - b) enhance SIMD instructions

2. **Compress** Aggregates of Front Stack:
   - a) single value kept in **register**

3. **Reduce** the size of Back Stack:
   - a) single value per **slide**

**Reduce cache footprint & operations: 33% improvement**
Bulk Insertion & SIMD: vectorising computation

- **SIMD-parallel versions** of aggregate functions
- **Vectorise** the computational intensive parts:
  - Insertion: **pre-aggregate** values upon bulk insert
  - Eviction: parallel scan for **SoE**
  - Flip Phase: vectorised-version for **Two-Stacks**

```c
_mm256_min_epi32(vec_input[i], vec_input[i+1]);
```

Parallel Processing within the slide (>1): 4.2× speedup
Experimental Setup

- **Hardware**
  - Single-threaded execution on a Intel Xeon CPU E5-2640 v3 2.60 GHz
    - 20MB LLC cache
  - 64 GB RAM
  - gcc version 7.3.0

- **Invertible and non-invertible functions** (min/max, sum, average,...)
  - Throughput comparison

- **Benchmarks & Real-world Dataset**

  Google Cluster Data
  144M jobs events from Google infrastructure
The Effect of Optimisations on Sliding Windows

MIN - Window Size 1024

Throughput (10^6 tuples/s)

Window Slide (tuples)

Non-Incr

Two-Stacks
For intermediate slides: up to $8\times$ throughput gains

When slide = $\frac{1}{2}$ size: cheaper to recompute
Real-world Scenario: Variable Windows

spiky trace:
- 0-100K events/sec
- variable windows
- stress implementations

AVG, Window Size 64 sec

Throughput (10^6 tuples/s)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>4</th>
<th>8</th>
<th>16</th>
<th>32</th>
<th>64</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-Incr</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>64</td>
</tr>
<tr>
<td>SoE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Two-Stacks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Window Slide (sec)
Robust performance: cope with variations in the workload!
CPU-efficient profile for tumbling windows

MIN, Tumbling Window

Performance dominated:
- strongly by Back-End (memory-bandwidth)
- less by Retiring
CPU-efficient profile for tumbling windows

Performance dominated:
- strongly by Back-End (memory-bandwidth)
- less by Retiring

CPU-efficient implementation!
Future Work

- Provide an efficient operator for holistic functions
- Highly efficient streaming operators (CPUs, FPGAs, GPUs)
- Just-in-time generation of platform-specific code
- Hardware-oblivious primitives that optimise computation based on windows
- Compilation-based techniques to deal with memory stalls
  - data in CPU registers
Summary

**Problem Statement:**
How to compute both invertible and non-invertible functions with high performance for variable slides?

**Solution:**
- **Bridge** the gap between *sliding* and *tumbling* windows with a single unified operator
- Highly **CPU- & Work- efficient** streaming aggregation
- Extend Two-Stacks algorithm
- Introduce **parallel processing** within a slide (>1)

Thank You!
Questions?

Georgios Theodorakis
http://lsds.doc.ic.ac.uk
<grt17@imperial.ac.uk>
Integration with SABER

AVG - Window Size 1024

Throughput (10^6 tuples/s)

Window Slide

0 1 32 64 128 256 512 1024

SABER
SABER(direct buffers)
SABER(simd)

3x
Integration with SABER

MIN - Window Size 1024

- SABER
- SABER (direct buffers)
- SABER (simd)

Throughput (10^6 tuples/s)

Window Slide

Throughput:
- 78.7x increase
- 3.25x increase
Stack RLE Compression

Example: Report the minimum requested CPU utilisation of submitted tasks

```sql
select timestamp, MIN(cpu)
from TaskEvents
```

Exploitable pattern on the Back Stack:
High probability that the aggregate value is equal or smaller to the value beneath it

Run-length encoding:
- Extra data dependencies (9-18% worse performance)