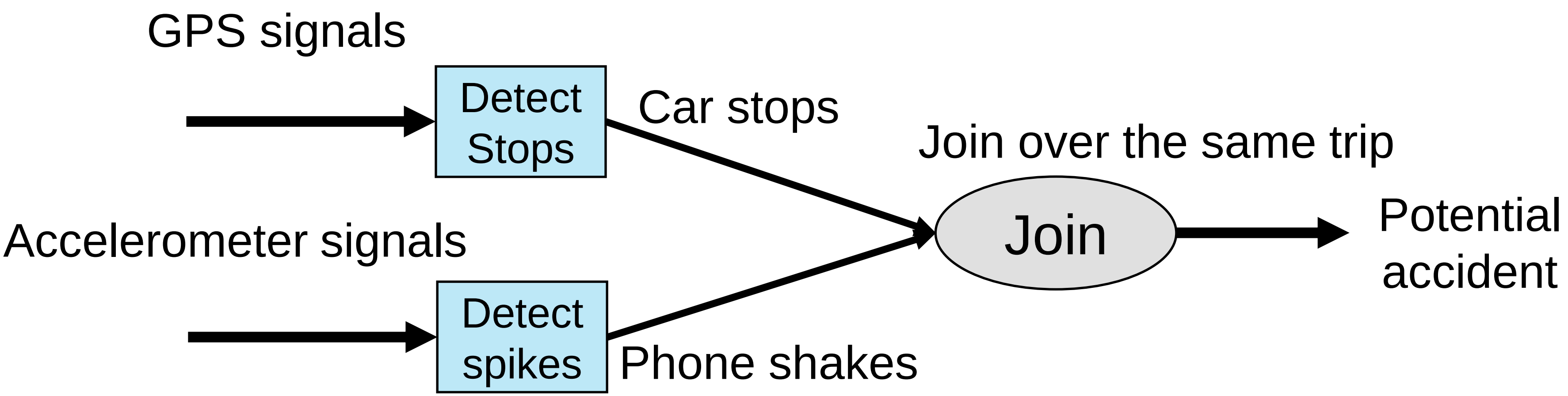
Lazy Evaluation of Sliding Window Join on Modern Multicores

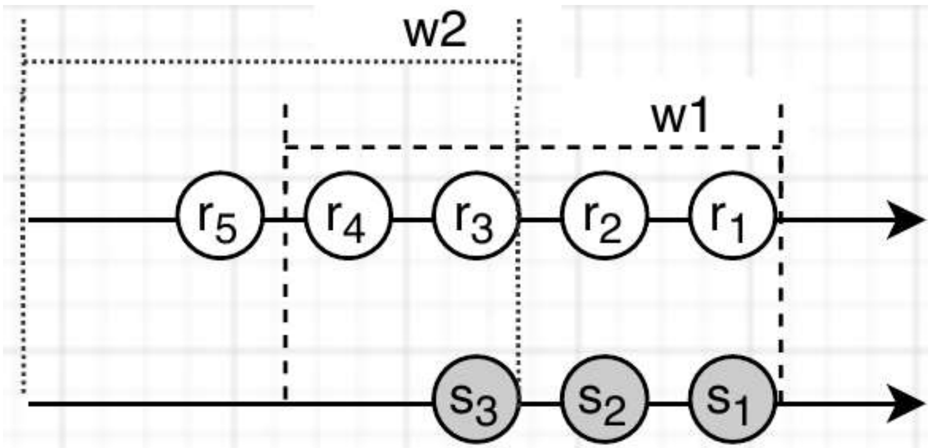
Shuhao Zhang

Stream Join



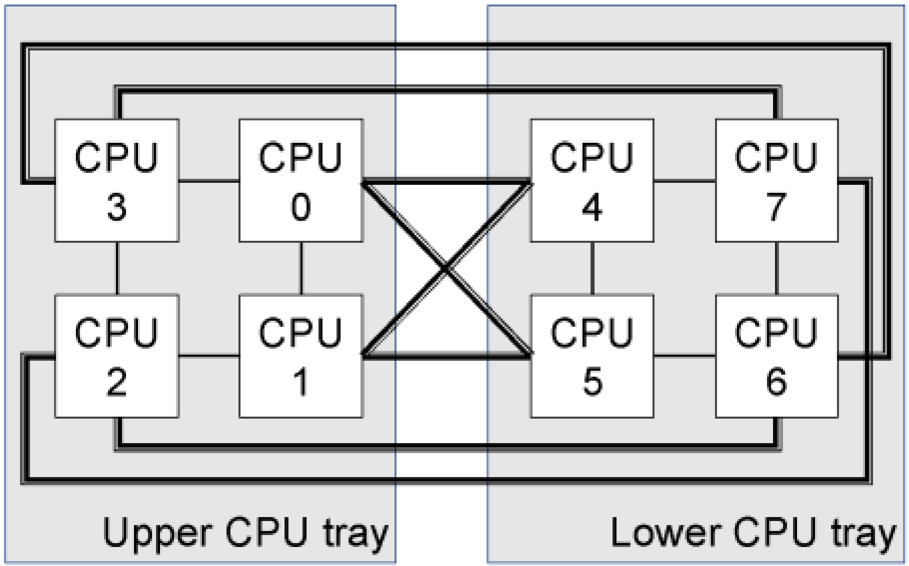
Credit: How Uber Detects on Trip Car Crashes – Nicolas Anderson & Jin Yang, Uber (Flink Forward, Oct, 2019)

Background: Sliding Window Join



- Sliding window join: joining over subsets (e.g., w1) of two input stream.
- Sliding window join is costly and significant efforts have been spent on accelerating it utilizing hardware parallelism.

Background: Modern Multicore Processors



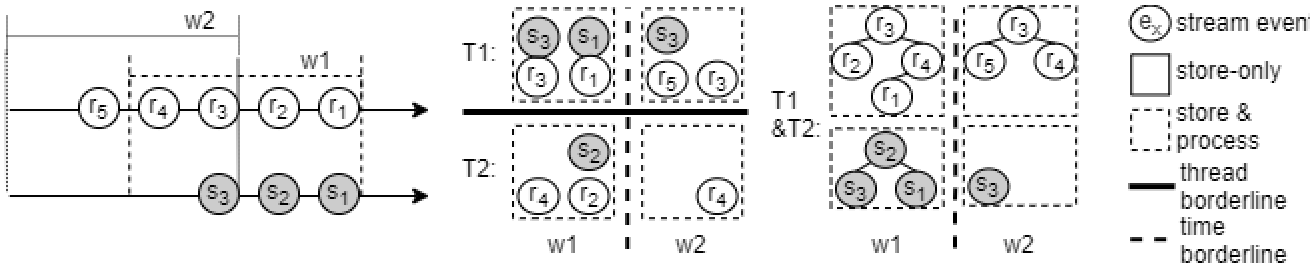
HUAWEI KunLun Server;
8 * 18 Cores (w/o HyperThreading)

Research Goal

Goal: achieve ultra-fast sliding-window join processing by better utilizing modern multicore processors

- There is a tradeoff between maximizing execution parallelism and maximizing sharing computing among windows

Prior Work



(a) HSJoin/SplitJoin/Bistream (b) IBWJ

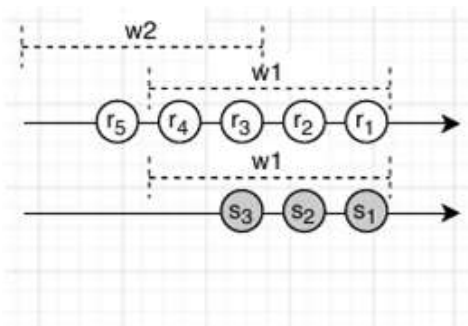
- They are all eager incremental Single-Window-based approach:
  - Frequent state updates involving significant commu./synch. overhead
  - Eager processing strategy involves severe cache thrashing issues

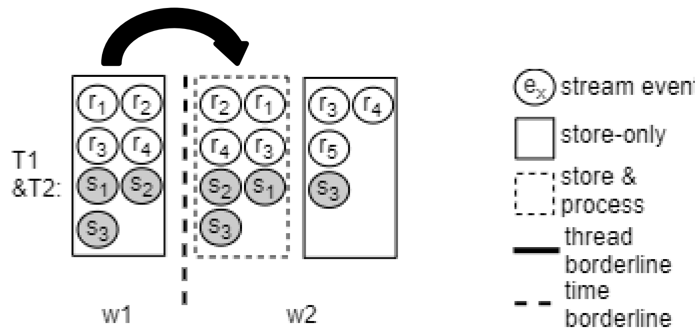
Existing Solutions Revisited

- Significant overhead due to windowing update
- Severe cache thrashing issues

A new solution is required!

Our Proposal: Lazy Window Join (LWJ)





(c) Lazy Window Join (LWJ)

- We adopt lazy incremental Multi-Window-based approach:
  - Wisely utilize hardware resource for each window with complete set of tuples
  - Efficiently reuse intermediate results to minimize recomputing overhead

Design Overview

- LWJ is achieved by two relatively independent components
  a. Intra-Window Join Processor
     i. Maximize computing efficiency of each window
     ii. With a cost-model to guide the parameter configurations
  b. Sliding Window Controller
     i. Minimize overall computing workloads by exploring shared-workloads
     ii. With a cost-model to guide number of windows

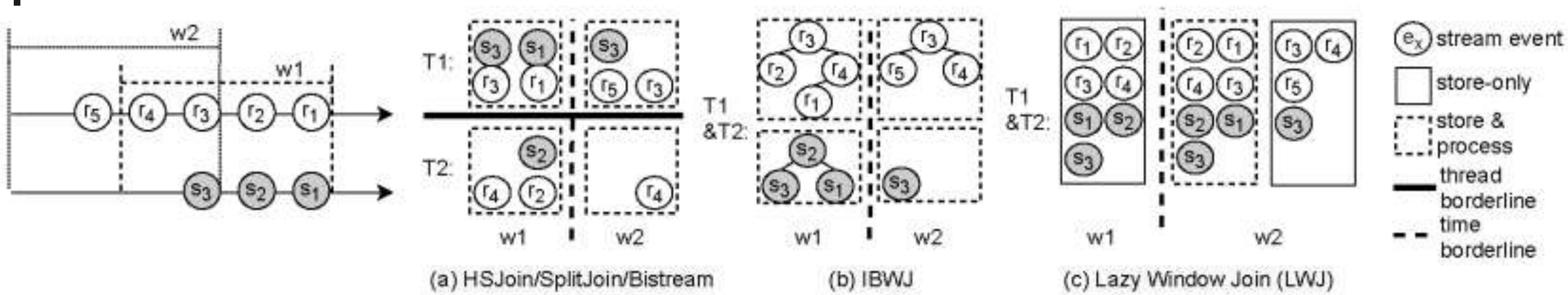
Intra-Window Join Processor

Key: Applying highly efficient relational join algorithm (e.g., radix parallel join) in processing each window

Sliding Window Controller

Key: Applying efficient lazily update of intermediate states to support subsequent windows’ computing

Summary: Lazy Window Join (LWJ)



Algorithm	*Incremental Window Execution*	*Online Distribution Strategy (What)*	*Data Flow Mechanism (How)*	*Join Algorithm*
HSJoin	Tuple-wise incremental	Eagerly Partition-by-timestamp	Bi-directional flow	Stream Join
SplitJoin/ BiStream	Tuple-wise incremental	Eagerly Partition-by-timestamp	Broadcast	Stream Join
IBWJ	Task-wise incremental	Eagerly Partition-by-key	Shared Index	Stream Join
LWJ	Window-wise incremental	Non-Partition	Shared Input Array	Relational Join (e.g., Parallel Radix Join)

Some Remarks

IBWJ is nothing but a parallel-version of SHJ.

Remember to checkout the taxonomy

Plan (6 months)

- Based on AlianceDB*, implement HSJoin, SplitJoin and IBWJ. (2 months)
- Validate our hypotheses in slide 7. (0.5 month)
- Design Intra-Window Join Processor with cost-model to handle each window. (1 month)
- Design Sliding Window Controller with cost-model to handle window progress. (1 month)
- Put them together and evaluate the LWJ. (1.5 month)

*https://github.com/ShuhaoZhangTony/SlidingWindowJoin