StreamBox: Modern Stream Processing on a Multicore Machine

Hongyu Miao and Heejin Park, Purdue ECE;
Myeongjae Jeon and Gennady Pekhimenko, Microsoft Research;
Kathryn S. McKinley, Google; Felix Xiaozhu Lin, Purdue ECE

http://xsel.rocks/p/streambox
High velocity of streaming data requires real-time processing
Streaming Pipeline

Infinite data stream

Pipeline

Input
Streaming Pipeline

Infinite data stream

Input

Pipeline

Epoch

Epoch

Epoch
Streaming Pipeline

Infinite data stream

Input

Transform 0

Epoch

Epoch

Epoch
Streaming Pipeline

Infinite data stream

Input

Transform 0

Transform 1

Transform 2

Epoch

Epoch

Epoch

Epoch
Streaming Pipeline

Infinite data stream

Pipeline

Input

Transform 0

Transform 1

Transform 2

Output
Why is it hard?
Records arrive out-of-order
Why is it hard?

Records arrive out-of-order

High Performance on Multicore
  • Data parallelism
  • Pipeline parallelism
  • Memory locality

Infinite data stream

Input

Transform 0
  Epoch

Transform 1
  Epoch

Transform 2
  Epoch

Output

Core 0
  960KB
  3584KB

Core 1
  960KB
  3584KB

Core 2
  960KB
  3584KB

... Core 13
  960KB
  3584KB

35MB L3

NUMA 0

NUMA 1

NUMA 2

NUMA 3

Intel Xeon E7-4830 v4
Prior work

Out-of-order processing within epochs

Processes only one epoch in each transform at a time
Prior work

Out-of-order processing within epochs

Processes only one epoch in each transform at a time

Pipeline
Prior work

Out-of-order processing within epochs

Processes only one epoch in each transform at a time
StreamBox insight

Out-of-order processing across epochs

Process all epochs in all transforms in parallel
Prior work vs. StreamBox

Processes only one epoch in each transform at a time

Process all epochs in all transforms in parallel

StreamBox: High pipeline and data parallel processing system
Result: StreamBox vs. existing systems on multicore

High throughput & utilization of multicore hardware

Throughput KRec/s

# Cores

StreamBox
Spark Streaming
Beam
Roadmap

Background

Stream pipeline, streaming data, window, watermark, and epoch

StreamBox Design

• Invariants to guarantee correctness
• Out-of-order epoch processing

Evaluation
Streaming pipeline for data analytics

**Transform** a computation that consumes and produces streams

**Pipeline** a dataflow graph of transforms

Ingress → Transform 1 → Transform 2 → Egress

Group by word → Count word occurrences

**A Simple WordCount Pipeline**
Stream records = data + event time

Records arrive out of order
  • Records travel diverse network paths
  • Computations execute at different rates
Window

A temporal processing scope of records
Chopping up infinite data into finite pieces along temporal boundaries
Transforms do computation based on windows

Infinite input stream

1:13 1:09 1:11 1:08

Window 1:00 – 1:05

Event Time

1:02 1:03
Window

A *temporal processing* scope of records

Infinite input stream

Windows by event time

1:00 – 1:05
Window

A **temporal processing** scope of records

**Infinite input stream**

<table>
<thead>
<tr>
<th>1:13</th>
<th>1:09</th>
<th>1:11</th>
<th>1:08</th>
<th>1:03</th>
</tr>
</thead>
<tbody>
<tr>
<td>🟢</td>
<td>🟢</td>
<td>🟢</td>
<td>🟢</td>
<td>🟢</td>
</tr>
</tbody>
</table>

**Windows by event time**

1:00 – 1:05

1:02
Window

A temporal processing scope of records

Infinite input stream

Windows by event time

1:00 – 1:05

1:03
1:02

1:08
1:11
1:09
1:13
Window

A temporal processing scope of records

Infinite input stream

1:13
1:09
1:11

Windows by event time

1:08
1:02
1:03

1:05 – 1:10
1:00 – 1:05
Window

A temporal processing scope of records

Infinite input stream

Windows by event time

1:10 – 1:15
1:05 – 1:10
1:00 – 1:05
Window

A temporal processing scope of records
Window

A **temporal processing** scope of records

Infinite input stream

Windows by event time

1:13  1:11
1:10 – 1:15

1:09  1:08
1:05 – 1:10

1:03  1:02
1:00 – 1:05
Out-of-order records

Infinite input stream

Windows by event time

1:13
1:09
1:11

1:08
1:05 – 1:10
When a window is complete?

Infinite input stream

Windows by event time

1:10 – 1:15

1:05 – 1:10
Watermark

Input completeness indicated by data source

**Watermark X** all input data with event times less than X have arrived
Handling out-of-order with watermarks

Infinite input stream

Watermark 1:10

1:13
1:09
1:11
1:08

Windows by event time

1:05 – 1:10

Watermark 1:05
Handling out-of-order with watermarks

Infinite input stream

Watermark 1:10

1:13

1:09

1:11

Watermark 1:05

Windows by event time

1:08

1:05 – 1:10

1:05 – 1:10
Handling out-of-order with watermarks

Infinite input stream

Watermark 1:10

Windows by event time

1:11

1:10 – 1:15

1:08

1:05 – 1:10

Watermark 1:05
Handling out-of-order with watermarks

Infinite input stream

Windows by event time

Watermark 1:10

1:13

1:11

1:10 – 1:15

1:09
1:08

1:05 – 1:10

Watermark 1:05
Handling out-of-order with watermarks
Epoch

A set of records arriving between two watermarks

A window may span multiple epochs
Roadmap

Background

StreamBox Design
  • Invariants to guarantee correctness
  • Out-of-order epoch processing

Evaluation
Stream processing engines

Most of stream engines optimize for a distributed system
  • Neglected efficient multicore implementation
  • Assume a single machine incapable of handling stream data
Goal A stream engine for multicore

Multicore hardware with
  • High throughput I/O
  • Terabyte DRAMs
  • A large number of cores

A stream engine for multicore
  • **Correctness** respect dependences with minimal synchronization
  • **Dynamic parallelism** processes any records in any epochs
  • **Target** throughput & latency
Challenges

**Correctness**
- Guarantee watermark semantics by meeting two invariants

**Throughput**
- Never stall the pipeline

**Latency**
- Do not relax the watermark
- Dynamically adjust parallelism to relieve bottlenecks
Invariant 1  Watermark ordering

Transforms consume watermarks in order
Transforms consume all records in an epoch before consuming the watermark
Invariant 2  Respect epoch boundaries

Once a transform assigns a record an epoch, the record never changes epochs.
Invariant 2  Respect epoch boundaries

Once a transform assigns a record an epoch, the record never changes epochs.
Invariant 2  Respect epoch boundaries

What if a record changes to a later epoch?

Violate watermark guarantee!
Invariant 2  Respect epoch boundaries

What if records change to an earlier epoch?

Relax watermark, and delay window completion!
Our solution: Cascading containers

Each cascading container
- Corresponds to an epoch
- Tracks an epoch state and the relationship between records and the watermark
- Orchestrates worker threads to consume watermarks and records
Each transform has multiple containers

A transform has multiple epochs
Each epoch corresponds to a container
Link each container to a downstream container defined by the transform
Records/watermarks flow through the pipeline by following the links

Meets invariant 2: records respect epoch boundary
Avoids relaxing watermark
A watermark will be processed after all records within the container have been processed

Guarantees the invariant 1: watermark ordering
Watermarks will be processed in order

Guarantees the invariant 1: watermark ordering
All records in all containers can be processed in parallel

Avoids stalling pipeline

Transform 0

Transform 1
Big picture

A pipeline: multiple transforms
- Containers form a network
- Records/watermarks flow through the links

High parallel pipeline
- Guarantees watermark semantic
- Avoids stalling pipeline (for throughput)
- Avoids relaxing watermark (for latency)
Other key optimizations

Organizing records into bundles
  • Minimize synchronization

Multi-input transforms
  • Defer container ordering in downstream

Pipeline scheduling
  • Prioritize externalization to minimize latency

Pipeline state management
  • Target NUMA-awareness and coarse-grained allocation
StreamBox implementation

Built from scratch in 22K SLoC of C++11
  • Supported transforms: Windowing, GroupBy, Aggregation, Mapper, Reducer, Temporal Join, Grep...
  • Source code @ http://xsel.rocks/p/streambox

C++ libraries
  • Intel TBB, Facebook folly, jemalloc, boost...

Concurrent hash tables
  • Wrapped TBB’s concurrent hash map
StreamBox implementation

Benchmarks:
- Windowed grep
- Word count
- Counting distinct URLs
- Network latency monitoring
- Tweets sentiment analysis

Machine configurations:

**CM12**
- 6 cores
- 6 cores
- 256GB DRAM

**CM56**
- 14 cores
- 14 cores
- 14 cores
- 14 cores
- 256GB DRAM
Roadmap

Background
StreamBox Design
Evaluation
Evaluation

Throughput and scalability

Comparison with existing stream engines

Handling out-of-order input streaming data

Epoch parallelism effectiveness
Good throughput and scalability
Good throughput and scalability
Good throughput and scalability
Good throughput and scalability
StreamBox vs. existing stream engines

StreamBox achieves significantly better throughput and scalability
Handling out-of-order records

StreamBox achieves good throughput even with lots of out-of-order records
Epoch parallelism is effective
Summary: StreamBox on multicores

Processes any records in any epochs in parallel by using all CPU cores

Achieves high throughput with low latency

- Millions records per second throughput, on a par with distributed engines on a cluster with a few hundreds of CPU cores
- Tens of milliseconds latency, 20x shorter than other large-scale engines

http://xsel.rocks/p/streambox