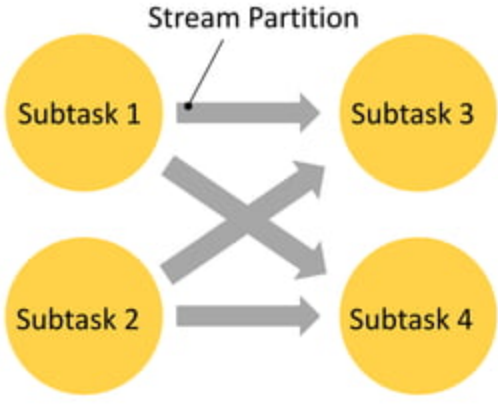
IMPROVING THROUGHPUT AND LATENCY WITH FLINK'S NETWORK STACK

NICO KRUBER

SOLUTION ARCHITECT / SOFTWARE ENGINEER @ DATA ARTISANS,
APACHE FLINK COMMITTER

dataArtisans

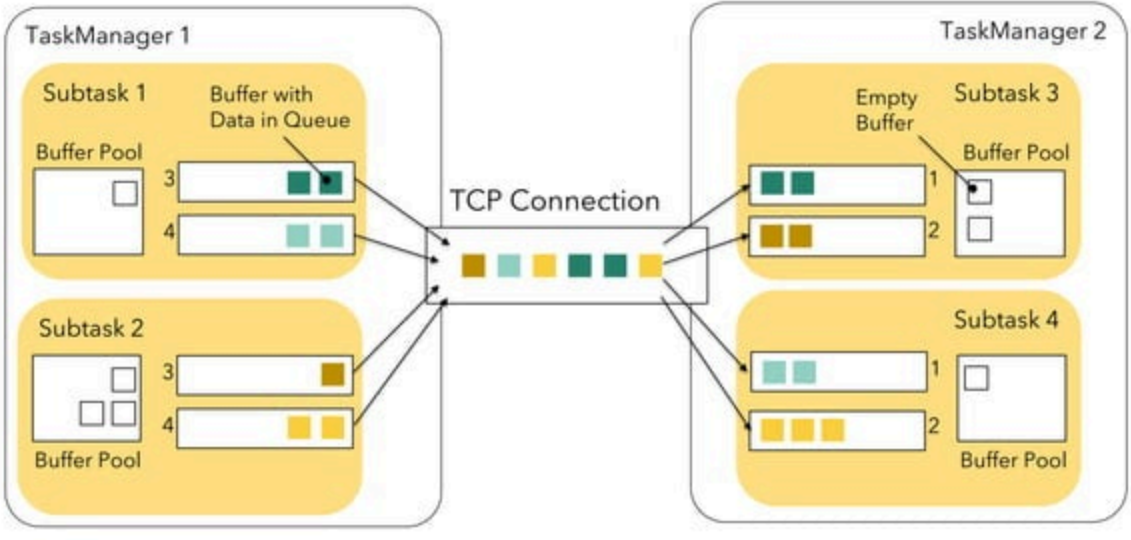
FLINK DATA TRANSPORT (LOGICAL)



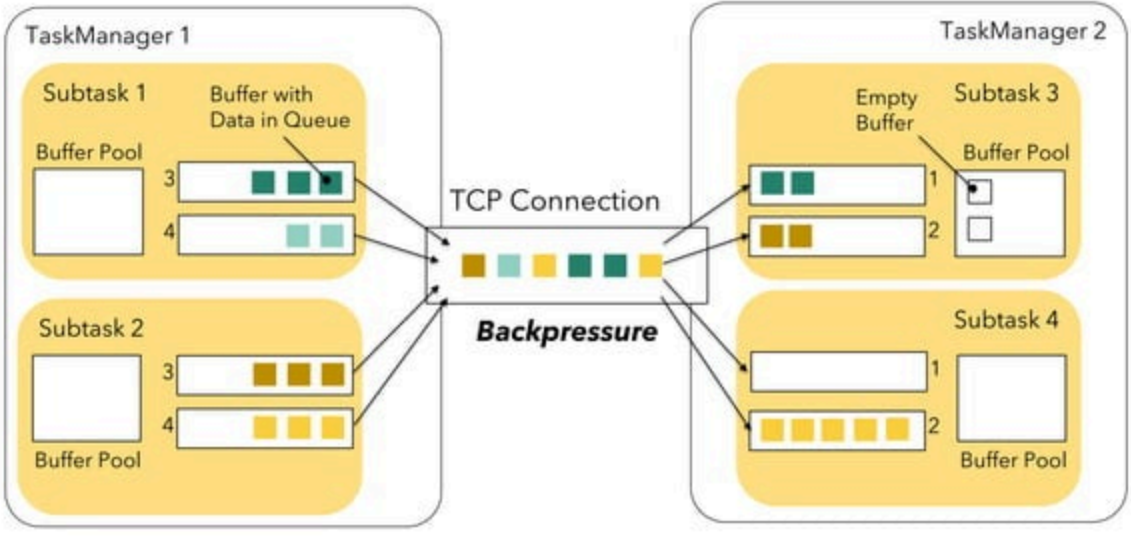
Abstraction over:

- Subtask output
  - pipelined-bounded
  - pipelined-unbounded
  - Blocking
- Scheduling type
  - all at once
  - next stage on complete output
  - next stage on first output
- Transport
  - high throughput via buffers
  - low latency via buffer timeout

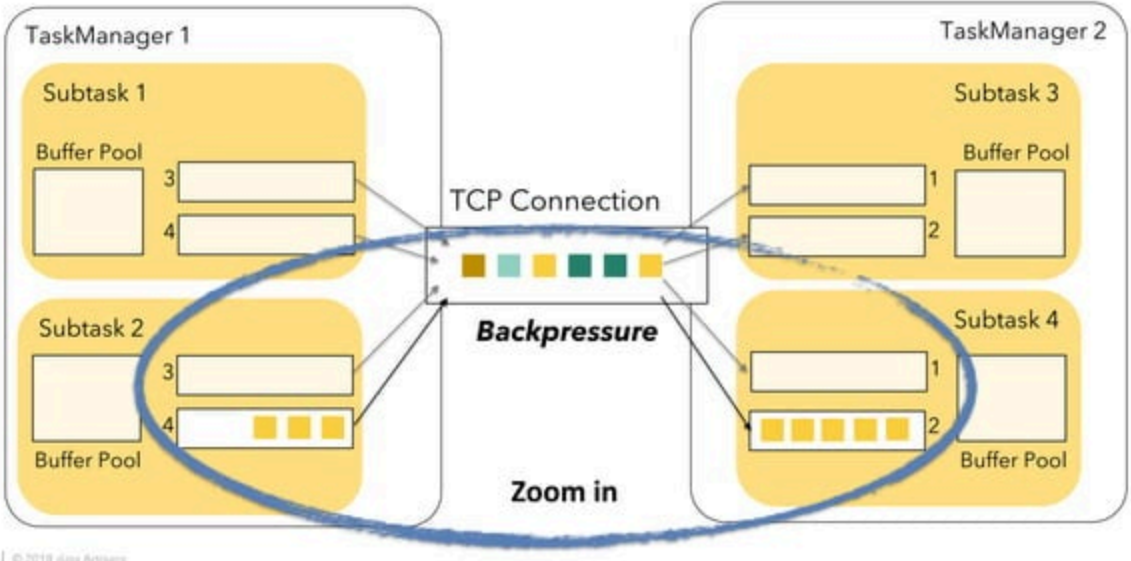
FLINK DATA TRANSPORT (PHYSICAL)



FLINK DATA TRANSPORT (PHYSICAL)

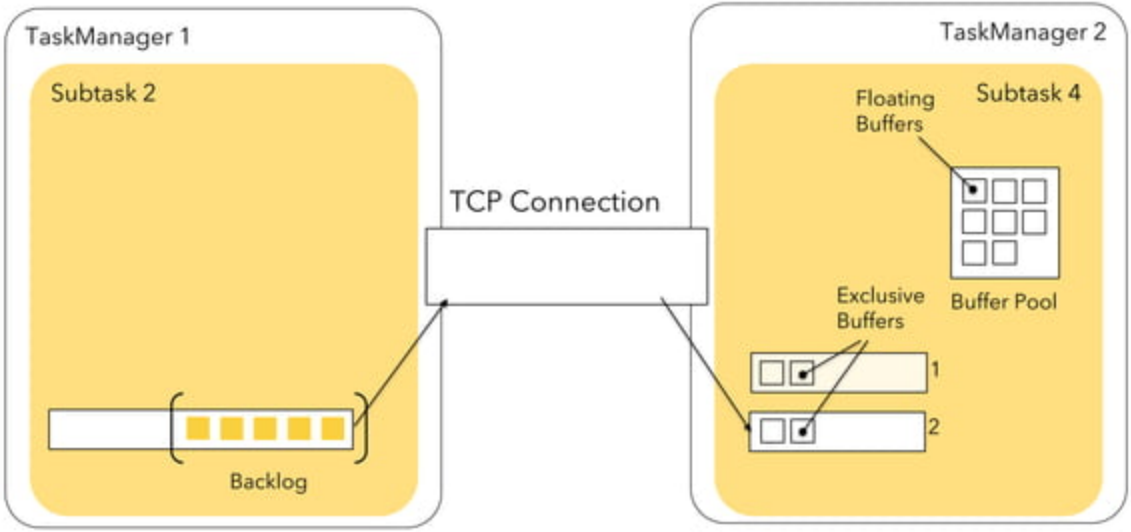


FLINK DATA TRANSPORT (PHYSICAL)



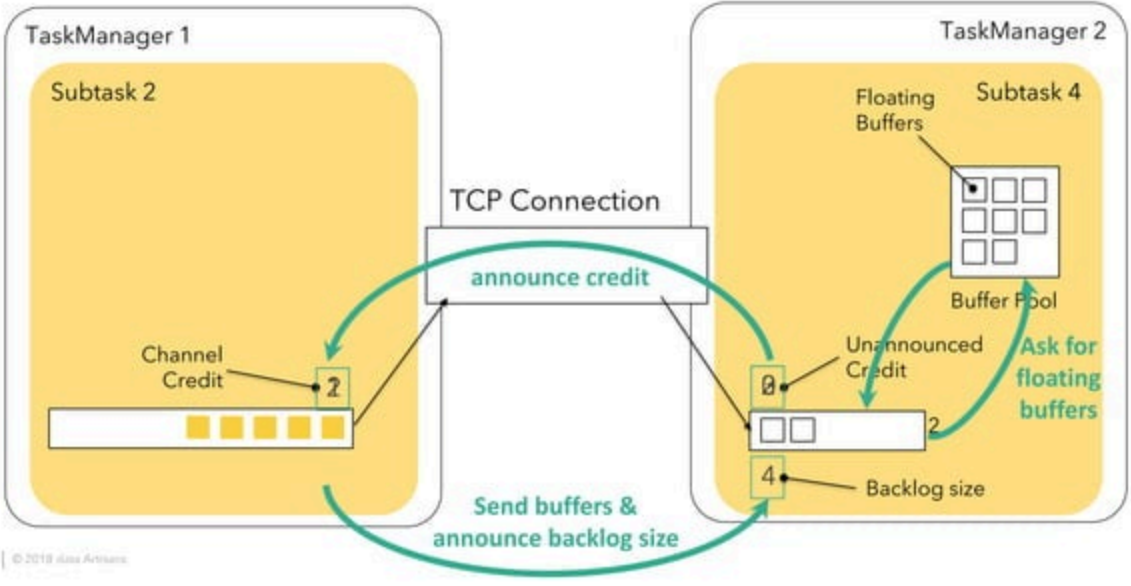
CREDIT-BASED FLOW CONTROL

CREDIT-BASED FLOW CONTROL (FLINK 1.5+)



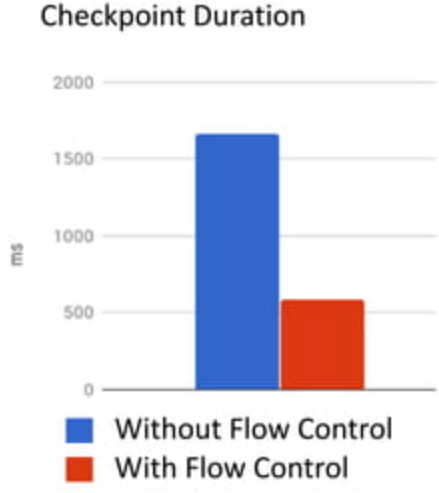
 |

CREDIT-BASED FLOW CONTROL (FLINK 1.5+)



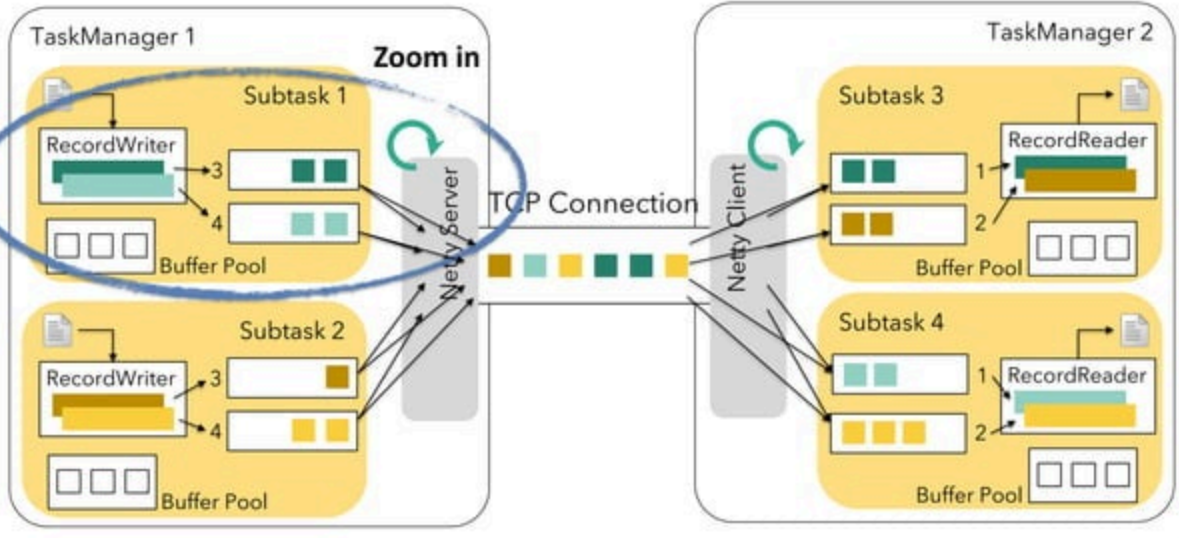
CREDIT-BASED FLOW CONTROL (FLINK 1.5+)

- Never blocks the TCP connection
- ➢ Better resource utilization with data skew in multiplexed connections
- Avoids overloading of slow receivers (direct control over amount of buffered data)
- ➢ Improves checkpoint alignment
- cost: additional announce messages (piggy-bagged), potential round-trip latency

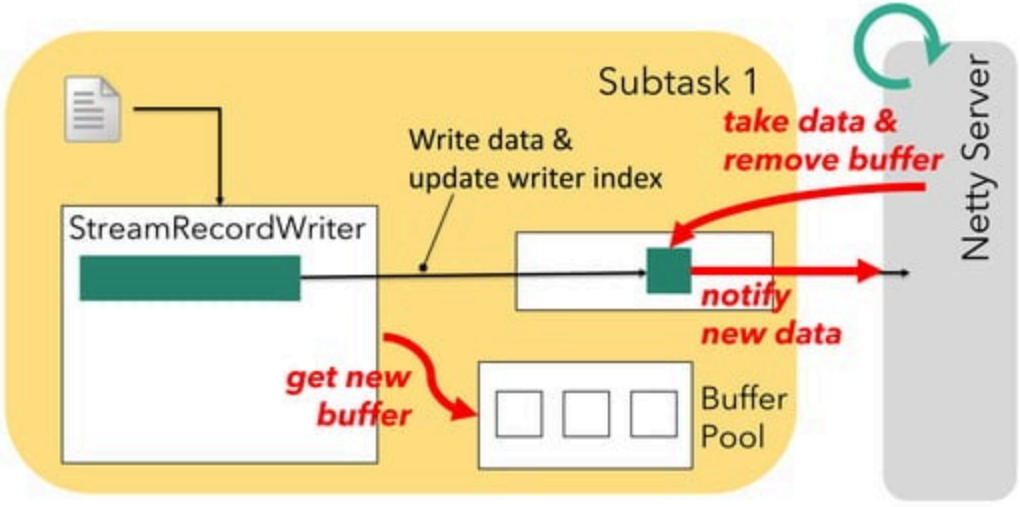


LOW LATENCY IMPROVEMENTS

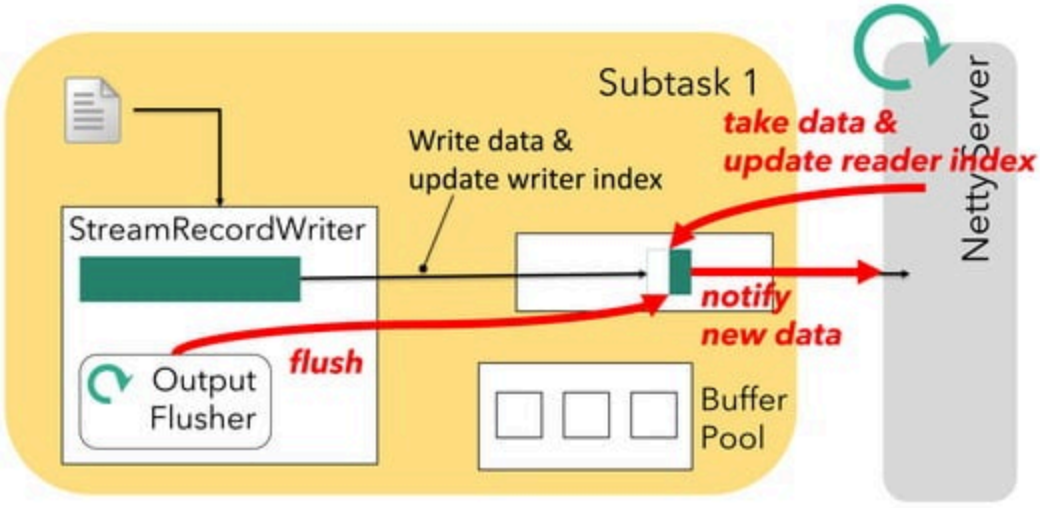
NETWORK STACK (EXTENDED)



FROM RECORD TO NETWORK

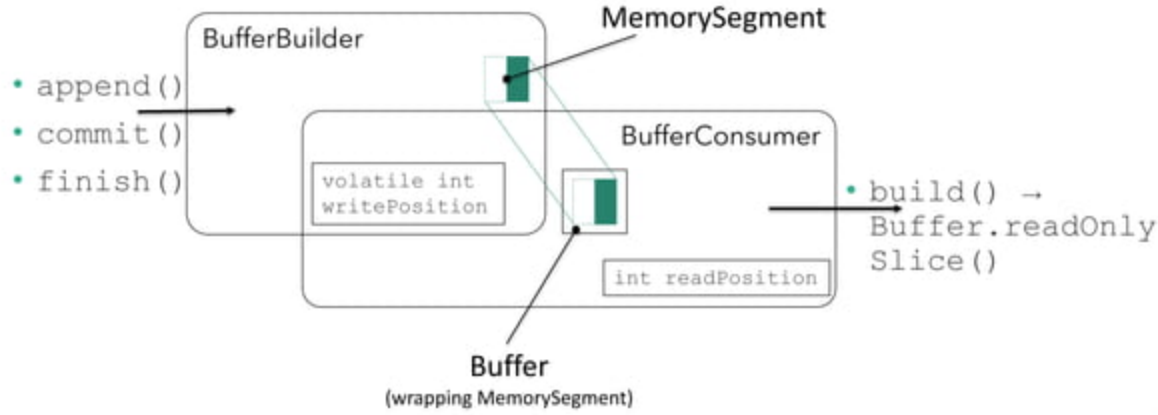


FROM RECORD TO NETWORK



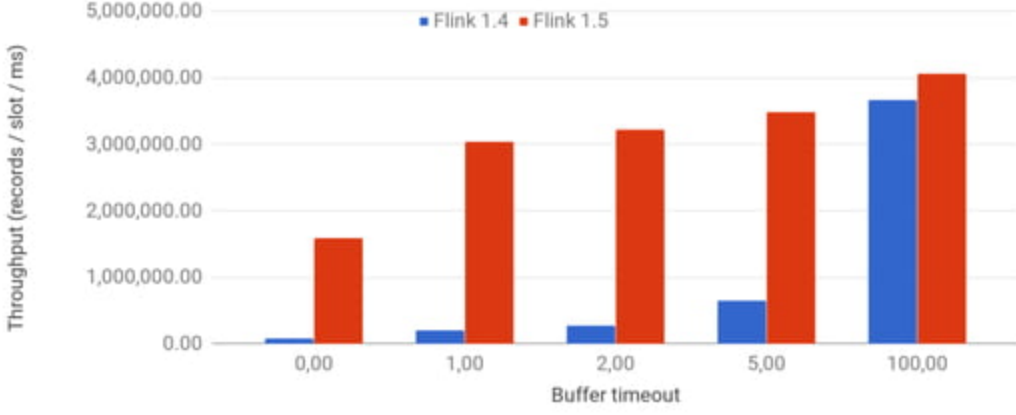
BUFFER BUILDER & CONSUMER

- Producer-Consumer structure with lightweight synchronization



LATENCY VS. THROUGHPUT

- low latency via buffer timeout
- high throughput through buffers



*100 nodes x 8 slots

CONNECTION TYPES

LOCAL VS. REMOTE CONNECTIONS

- Every (unchained) connection:
  - Requires serialization
  - Assembles serialized records into buffers
  - Forwards a buffer when it is full or the buffer timeout hit
- Remote connection:
  - Sent via multiplexed Netty TCP connections (one per pair of tasks and task managers)
  - As soon as a buffer is on the wire, it can be re-used
    - Allows credit-based flow control to control amount of buffered data
- Local connection:
  - Direct connection between sender and receiver: buffers are shared
    - No need for further flow control (buffered data = sender buffers)

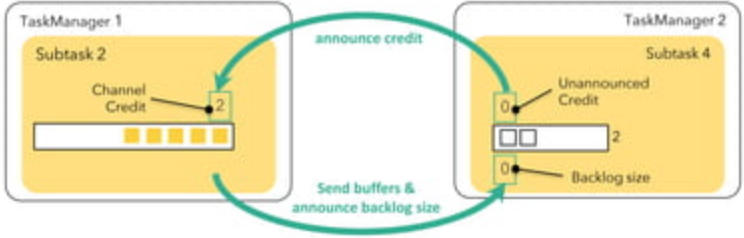
TUNING OPTIONS

CREDIT-BASED FLOW CONTROL

- `taskmanager.network.credit-model: true/false`
- `taskmanager.network.memory.buffers-per-channel: 2`
- `taskmanager.network.memory.floating-buffers-per-gate: 8`

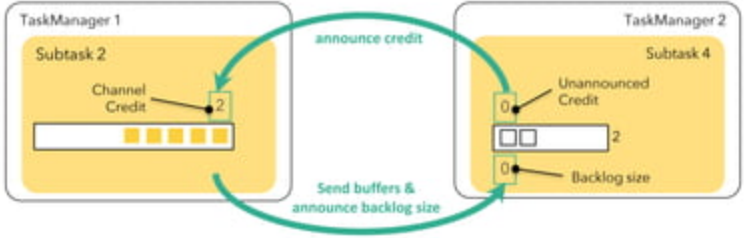
- Number of exclusive buffers should be enough to saturate the network for a full round-trip-time (2 x network latency)

➢ `#exclBuffers * segmentSize = round-trip-time * throughput`



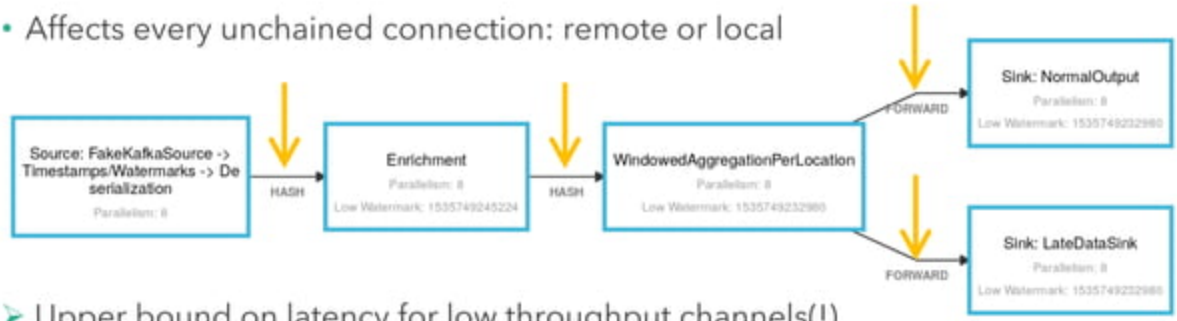
CREDIT-BASED FLOW CONTROL

- Number of exclusive buffers too high
  - higher number of required network buffers
  - buffering more during checkpoint alignment
  - BUT: faster ramp-up (before floating buffers kick in)
- Number of exclusive buffers too low
  - times of in-activity during ramp-up



BUFFER TIMEOUT

- `StreamExecutionEnvironment#setBufferTimeout()`
- Affects every unchained connection: remote or local



- ➢ Upper bound on latency for low throughput channels(!)
- ➢ Trade-off throughput vs. latency (see earlier)

NETWORK THREADS

- `netty.client.numThreads` (default: number of slots)
- `netty.server.numThreads` (default: number of slots)

- May become a bottleneck if thread(s) are overloaded
- BUT: may also become an overhead if too many

➢ Do your own benchmarks and verify for your job!

USE LINUX-NATIVE EPOLL (FLINK 1.6+)

- `taskmanager.network.netty.transport: AUTO | NIO | EPOLL`

- `EPOLL` may reduce the channel polling overhead between user space and kernel/system space
- There should be no downside in activating this or at least `AUTO`.

➢ Do your own benchmarks for your job!

- Please give feedback in FLINK-10177 so that we can decide whether to use `AUTO` by default.

METRICS

NETWORK STACK METRICS

- Backpressure monitor
  - Web/REST UI, /jobs/:jobid/vertices/:vertexid/backpressure)

- [input, output]QueueLength
- numRecords[In, Out]
- numBytesOut, numBytesIn[Local, Remote]
- numBuffersOut, numBuffersIn[Local, Remote] (Flink 1.5.3+, 1.6.1+)

LATENCY MARKERS

- `ExecutionConfig#setLatencyTrackingInterval()` (default: every 2s)

- Sources periodically emit a LatencyMarker with a timestamp
- These flow with the stream and properly queue behind records
- Latency markers bypass operators, e.g. windows
- Once received, they will be re-emitted onto a random output channel

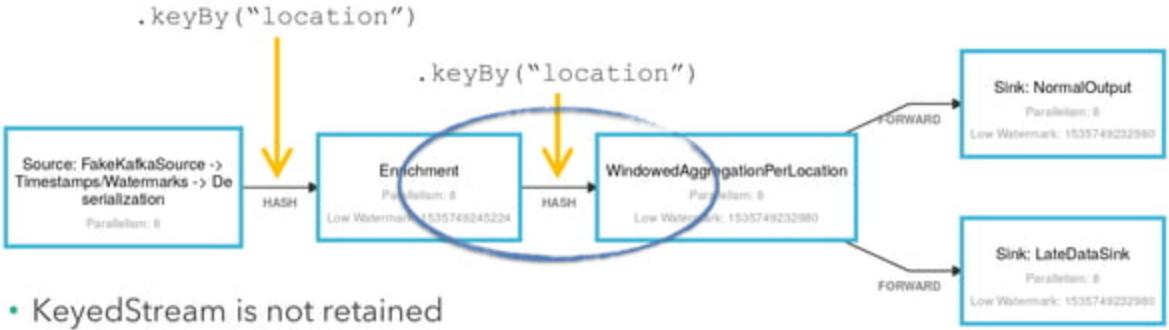
- We create one histogram per source ↔ operator pair (window size: 128)
- `source_id.<sourceId>.source_subtask_index.<subtaskIdx>.operator_id.<operatorId>.operator_subtask_index.<subtaskIdx>`

➢ 10 operators, parallelism 100 = 9 * 100 * 100 = 90,000 histograms!

https://ci.apache.org/projects/flink/flink-docs-stable/monitoring/metrics.html#latency-tracking

COMMON ANTIPATTERNS

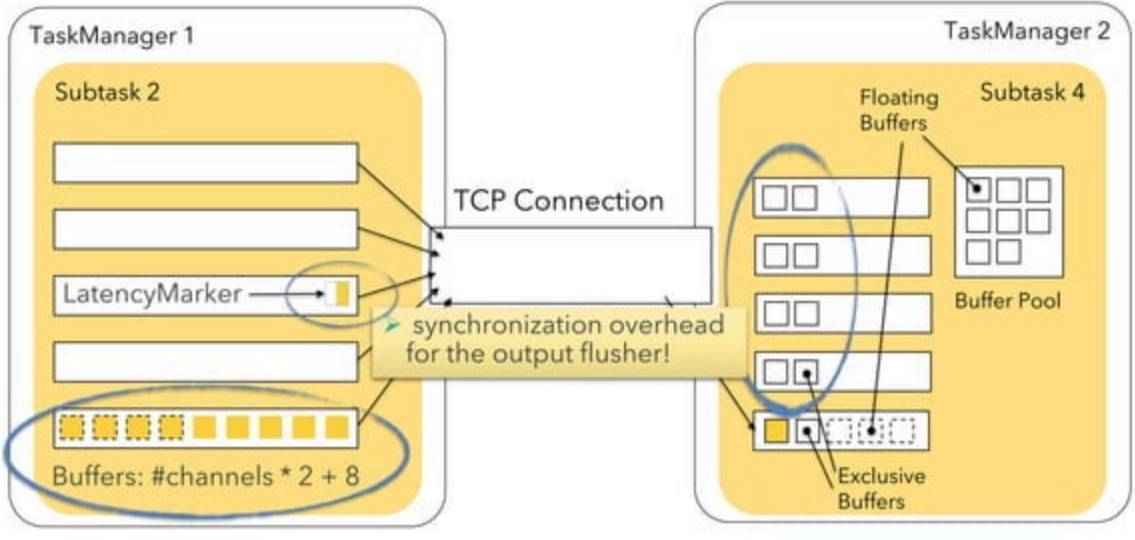
REPEATED KEYBY'S ON THE SAME KEY



- KeyedStream is not retained
  - UDF could have changed the key
- Additional keyBy() is necessary to gain access to keyed state, but:
  - Prevents chaining
  - Adds an additional shuffle

➢ `DataStreamUtils#reinterpretAsKeyedStream`

CREDIT-BASED FLOW CONTROL (FLINK 1.5+)



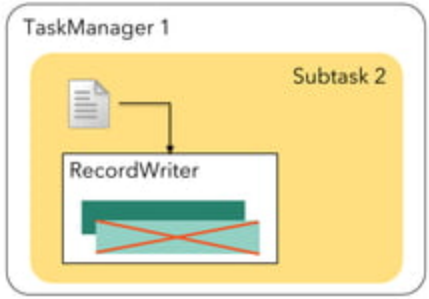
WHAT'S UP NEXT?

NETWORK SERIALIZATION STACK (FLINK 1.7?)

- Serialization for broadcasts once per record, not channel

- Only one intermediate serialization buffer (on heap)
  - ➢ significantly reduces the memory footprint

- see FLINK-9913



OPENSSL-BASED SSL ENGINE (FLINK 1.7?)

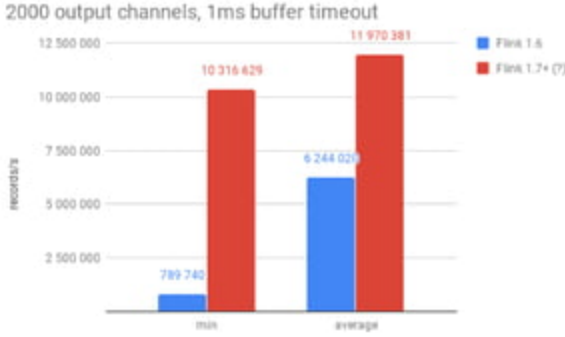
- Runs native code
- Uses advanced CPU instruction sets

➢ May reduce encryption/decryption overhead (needs verification)

- see FLINK-9816

MOVE OUTPUT FLUSHER TO NETTY

- Current implementation may have (GC) problems with many channels
  - schedule the output flusher inside the Netty event loop



- see [FLINK-8625]

THANK YOU!

@dataArtisans
@ApacheFlink

WE ARE HIRING
data-artisans.com/careers

dataArtisans