FLIP-183: Dynamic buffer size adjustment

Status

Discussion thread	https://lists.apache.org/thread.html/r9ddc3bee9649012f91e58c5a4a9a985d2256dc608e8bc37c69e39e52%40%3Cdev.flink.apache.org%3E
Vote thread	
JIRA	~~FLINK-23451~~ - FLIP-183: Dynamic buffer size adjustment (Buffer debloat) CLOSED
Release	1.14

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

Right now, the size of total network memory is set up statically in the configuration which makes the predictable level of maximum in-flight data but unpredictable time for which the in-flight data between two subtasks can be handled. In turn, it can lead to unpredictable/big delays between checkpoints and higher alignment time. Also worth noticing that it doesn't make sense to have so much data in buffers.

The idea is to change the implementation in such a way that holds the exact amount of data in buffers which can be handled for the configured amount of time.

This should improve aligned checkpoints time (fewer in-flight data to process before checkpoint can complete) and improve the behaviour and performance of unaligned checkpoints (fewer in-flight data that needs to be persisted in every unaligned checkpoint). Potentially this can be an alternative to using unaligned checkpoints.

Public Interfaces

Configuration

- The time in the input buffer

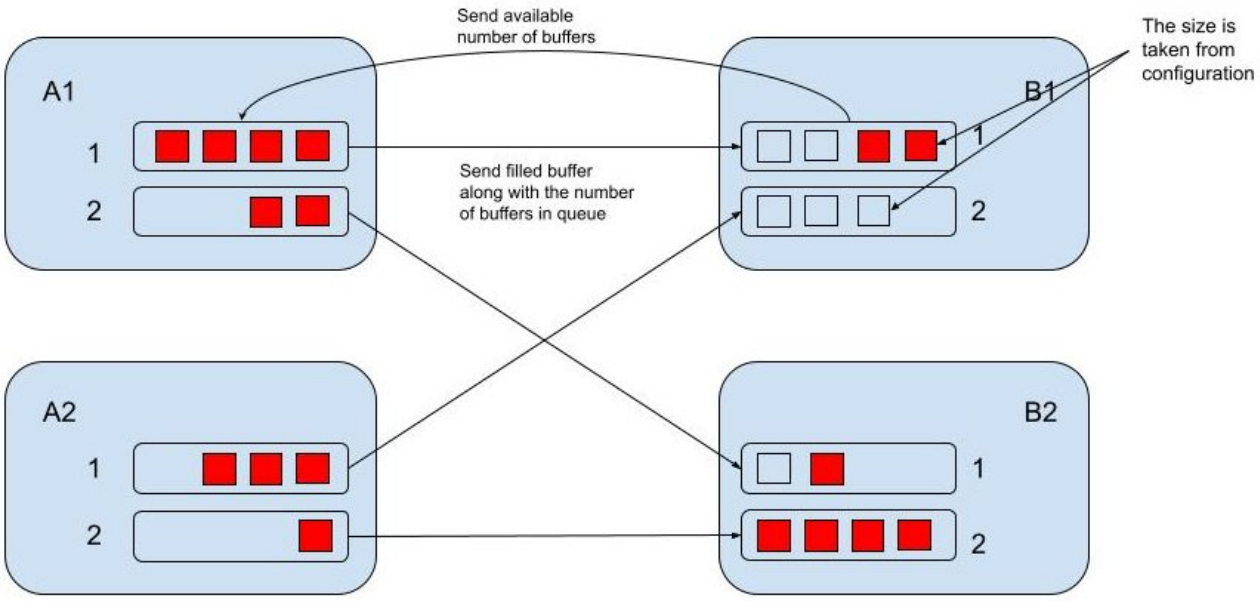
Metrics

- The total size of buffered in-flight data
- Total estimated time to process the data
- Actual value of the calculated dynamic buffer size

Web-UI

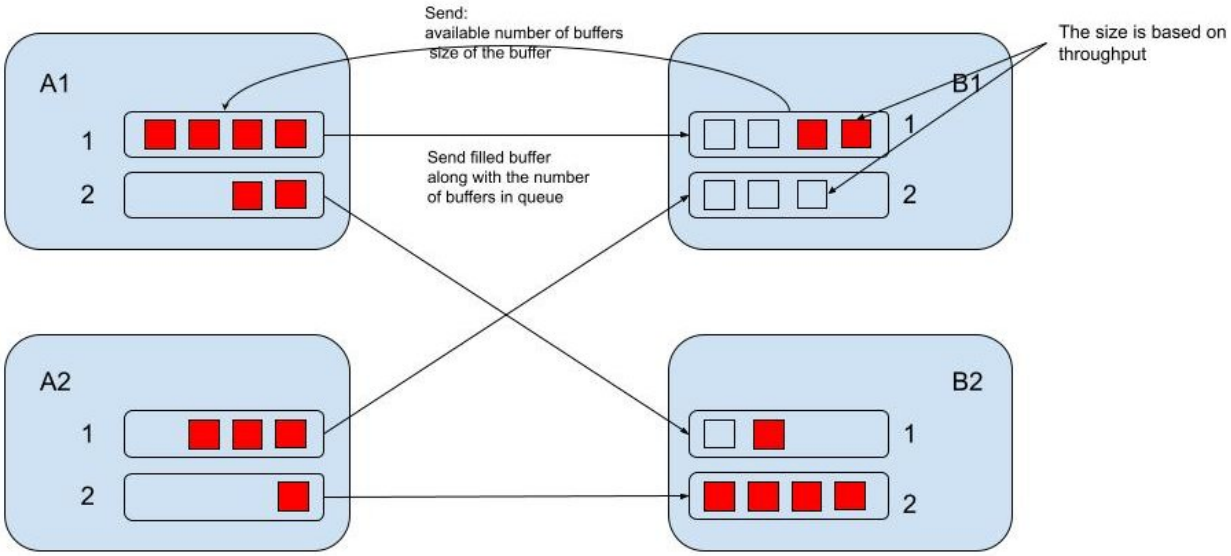
- The current size of the buffered in-flight data (upstream and downstream)
- Throughput and/or estimated time to process the in-flight data

Current implementation



- Subtasks B1 and B2 provide the number of available buffers to A1 and A2.
- Subtasks A1 and A2 can write to the buffers while they have at least one. And if they have credit from B task they can send the filled buffers to it.
- In the worst case, all buffers in the upstream and the downstream can be full which leads to unpredictable time for handle it.
- Total size of in-flight data in the worst case equals to (bufferSize * subPartitions * exclusiveBuffers + bufferSize * floatingBuffers) + (bufferSize * channel * exclusiveBuffers + bufferSize * floatingBuffers)

Proposed Changes



The most important difference here is that the one single buffer has a dynamic size while the number of buffers is constant(at least for the first implementation).

Main steps:

- The subtask calculate the throughput and then based on the configuration(timeInBufferQueue) calculate buffer size(bufferSize = throughput * timeInBufferQueue / totalNumberOfBuffers)
- The subtask observes the changes in the throughput and changes the information about buffer size during the whole life period of the task. But the subtask doesn't change the size physically. So the real size of the downstream buffer always remains as it was initially configured maximum size.
- The subtask sends the new buffer size and number of available buffers to the upstream to the corresponding subpartition.

- Upstream changes the size of the newly allocated buffer corresponding to the received information but the currently filled buffers send as is whatever their current size is. (it is possible because the downstream buffer size wasn't changed and it is able to handle any size up to maximum).
- Upstream sends the data and number of filled buffers to the downstream

Important notices for MVP:

- The size of each input channel is the same regardless of the difference in the throughput
- physically, the buffer is allocated the same memory but the size of it changes only logically(it is possible because the number of the buffers remains the same)
- Two/multiple input tasks can calculate the throughput independently for each gate
- UnionInputGate, perhaps, should be treated as one logical input(splitting throughput among each ‘SingleInputGate’)

Possible improvements in the future:

- Calculation of the throughput separately for each channel and distributing the buffers accordingly.
- Dynamic calculation of the number of buffers. This means that the buffers’ size and the number of buffers should be chosen automatically for the best performance.
- Real changing the buffer size(segment memory size). It requires avoiding fragmentation of the memory somehow.

Implementation Plan

Sending the buffer of the right size.

It is not enough to know just the number of available buffers (credits) for the downstream because the size of these buffers can be different. So we are proposing to resolve this problem in the following way: If the desirable downstream buffer size is changed then the upstream should send the already filled buffer as is but the buffer which will be allocated after the new buffer size information was received should be of the size not greater than the new one. (The request of the buffer should contain new input parameter like bufferSize)

Different subpartitions with different desired buffer size

If different downstream subtasks have different throughput and hence different desired buffer sizes, then a single upstream subtask has to support having two different subpartitions with different buffer sizes.

Measuring throughput

In the first implementation, throughput could be measured for the whole subtask. The throughput calculation should take into account the numbers of bytes that were handled, the backpressure time and ignore the idle time. The main idea is to keep the balance between idle and backpressure time, so if the backpressure time is high we should decrease the buffer size to provide the configured handling time and vice versa if the subtask is idle time that period should be ignored from calculating the throughput. Otherwise, in the case of network bottleneck, we might have ended up with a small buffer size that’s causing the bottleneck in the first place but we are not able to increase it due to idle time reducing throughput and lowering the buffer size.

Calculating the buffer size

To calculate the desired buffer size we need to take into account the throughput, configuration(timeInInputBuffer), and the actual number of buffers in use. It makes sense to use EMA for this calculation to smoothen out intermittent spikes.

The calculation based on the actual number of buffers in use helps to avoid problems with the data skew (when only a couple of channels out of thousands have any data). So the solution needs to reliably and efficiently calculate either the estimated or an average number of buffers in use.

Sending the buffer size to the upstream

Expanding the credit message to add the size of the buffer.

Metrics

The following metrics can be added:

- The total size of buffered in-flight data
- Total estimated time to process the data
- Actual value of the calculated dynamic buffer size

Web-UI

- The current size of the buffered in-flight data (upstream and downstream)
- Throughput and/or estimated time to process the in-flight data

The possible (not mutually exclusive) options:

- Add "performance" overlay on the job graph, that would display per task the metrics like: throughput, buffered data size
- In the per subtask view (current backpressure tab), add the same information but split per subtask.

Compatibility, Deprecation, and Migration Plan

Initially, there is no compatibility support requires because the solution is based on the existing configuration. But depends on the final implementation, there are some configurations like bufferSize or type of buffer that can become useless.

Benchmark Plan

Which parameters make sense to compare:

- The number of checkpoints per second/minute(We could modify the existing `UnalignedCheckpointTimeBenchmark` to test for `alignedCheckpoints` as well)
- The throughput with aligned checkpoint(the existing benchmarks are ok but perhaps we need something more that would take into account more parameters)
    - Different number of buffers
    - Different sizes of the buffer
- Flat rate job (with different throughputs and record sizes)
- Varying load job (for example some window operator that emits records in spikes)

It makes sense to add not only the benchmarks on the cluster but the microbenchmark as well where it is applicable.

Test Plan

Mostly, the existing regression tests will test these changes because there are not a lot of new features here, there are only several changes in existing ones.

Unit-tests for the calculation of the throughput are required along with the tests for changing buffer size based on the throughput.

Rejected Alternatives

There are no known alternatives yet