Metrics & Monitoring

Nico Kruber, Solutions Architect & Apache Flink committer

ververica

Metrics & Monitoring

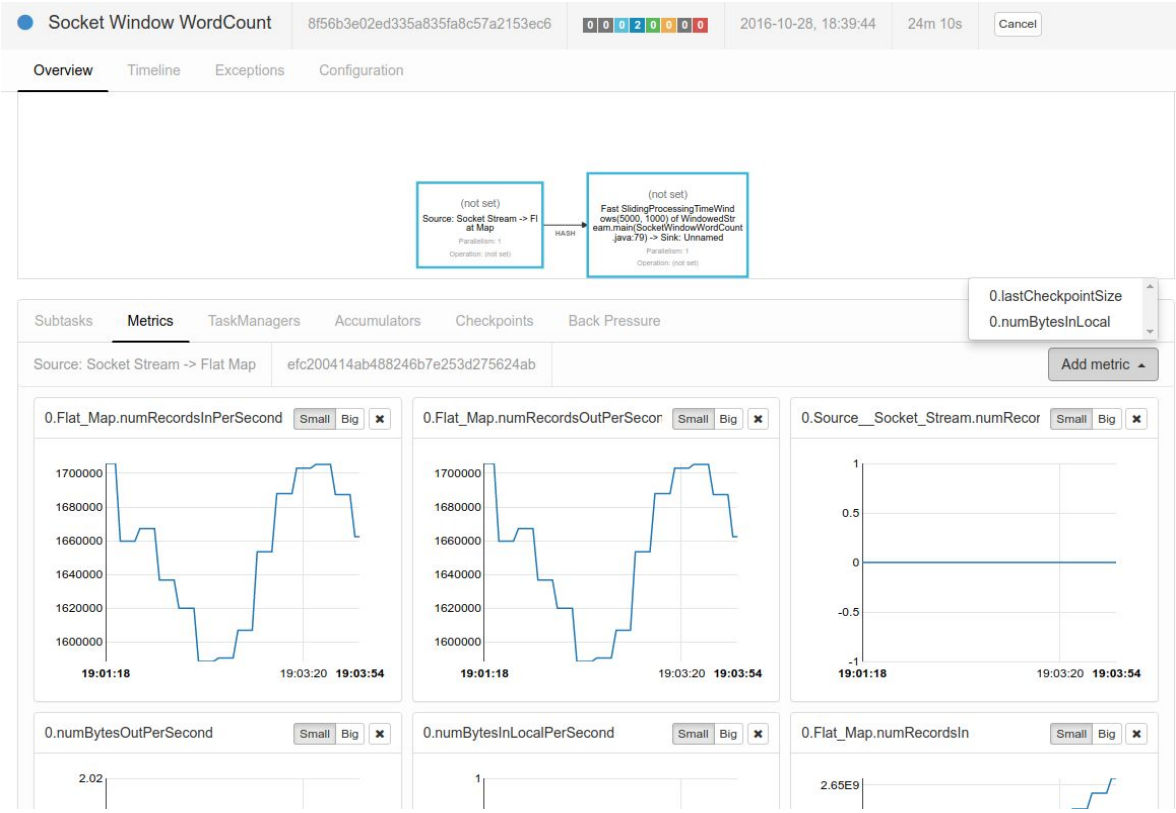
Agenda

- Flink's Metrics System "How"
  - `Metrics`
  - `MetricsReporter`
- Key Metrics for Continuous Monitoring *"What"*
  - Health
  - Throughput & Progress
  - Latency
- Key Metrics for Troubleshooting "What else"

Flink’s Metrics System

Metrics

- <identifier, measurement>
- Types
  - Counter
  - Meter (rate)
  - Histogram
  - Gauge (arbitrary value)



Example

```
public static class MyMap extends RichMapFunction<String, String> {
  private Counter count;

  @Override
  public void open(Configuration config) {
    count = getRuntimeContext()
      .getMetricGroup()
      .counter("numRecordsIn");
  }

  @Override
  public String map(String input) {
    count.inc();
    // return something
  }
}
```

Metrics

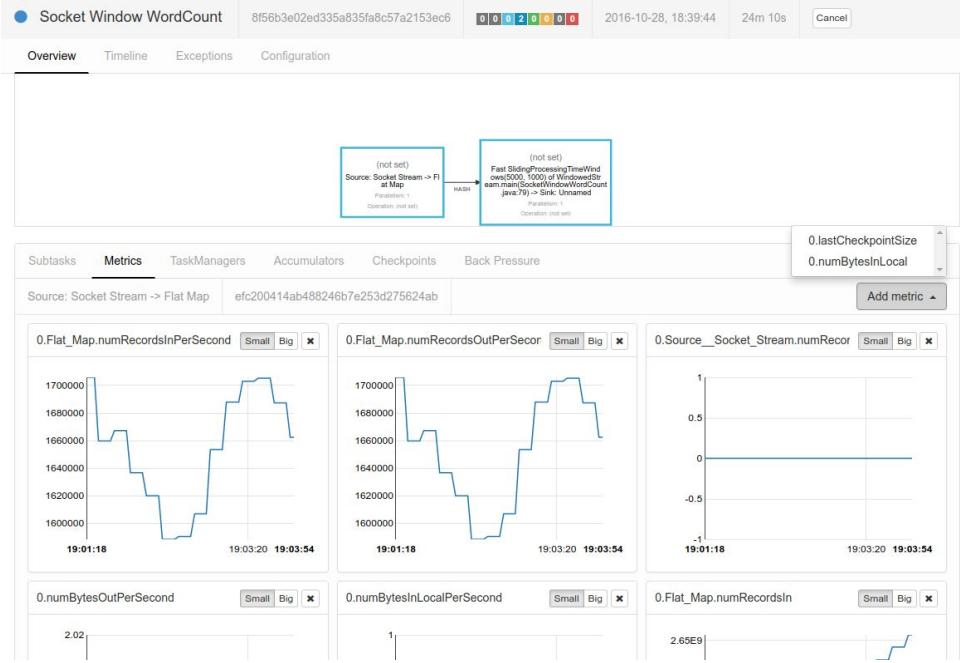
Scopes

- metrics scope to different levels of a Flink deployment
- the keys to attach to metrics in a certain scope can be configured
  - metrics.scope.jm: `<host>.jobmanager`
  - metrics.scope.task:

    `<host>.taskmanager.<tm_id>.<job_name>.<task_name>.<subtask_index>`
- Checkout https://ci.apache.org/projects/flink/flink-docs-release-1.9/monitoring/metrics.html#scope for details

Accessing Metrics

- WebUI → TaskMetrics
- REST API
- **`MetricsReporters`**



/jobs/<id>/metrics

/jobs/<id>/checkpoints

/jobs/<id>/vertices/<id>/metrics?get=0.numRecordsOutPerSecond

/taskmanagers/<id>/metrics?get=<metric>

Accessing Metrics

Metrics Reporters

- Datadog
- Ganglia
- Graphite
- JMX

- Prometheus
- StatsD
- SLF4J
- InfluxDB

Or write your own…

Accessing Metrics

A Simple Log4jReporter

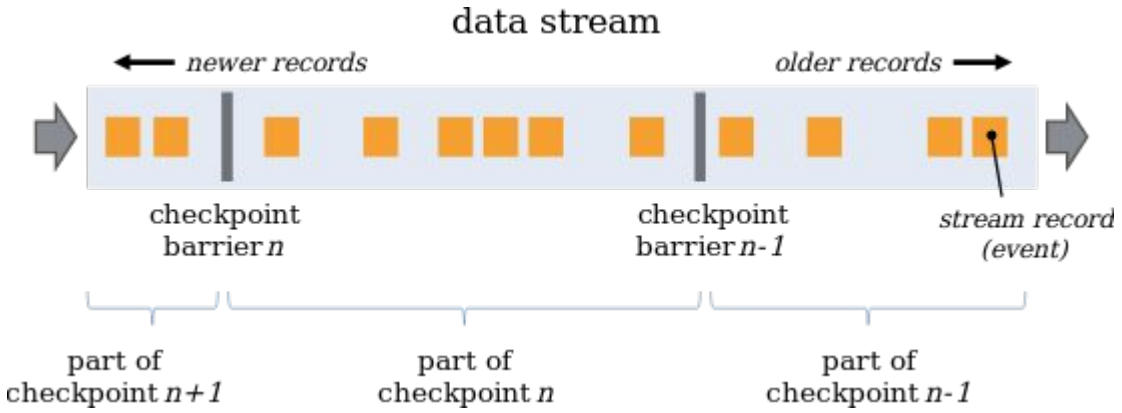
```
public static class Log4JReporter implements MetricReporter, Scheduled {
  private static final Logger LOG = LoggerFactory.getLogger(Log4jReporter.class);

  private final Map<Counter, String> counters = new ConcurrentHashMap<>();

  public void notifyOfAddedMetric(Metric metric, String metricName, MetricGroup group) {
    if (metric instanceof Counter) {
      counters.put((Counter) metric, group.getMetricIdentifier(metricName));
    }
  }

  public void notifyOfRemovedMetric(Metric metric, String metricName, MetricGroup group) {
    if (metric instanceof Counter) {
      counters.remove(metric);
    }
  }

  public void report() {
    for (Map.Entry<Counter, String> metric : counters.entrySet()) {
      LOG.info(metric.getValue() + ": " + metric.getKey());
    }
  }
}
```

Key Metrics for Continuous Monitoring

Key Metrics

General Health

- Is “RUNNING”?
  - `uptime`
  - `fullRestarts`
- Checkpointing Consistently?
  - `numberOfCompletedCheckpoints`
  - `numberOfFailedCheckpoints`
  - `lastCheckpointSize`



Key Metrics

Throughput & Progress

- Task & Operator Level Throughput
  - `numRecords(In|Out)PerSecond`
  - `numRecords(In|Out)`
- Progress & Event-Time Lag
  - `currentOutputWatermark`
- Keeping Up
  - `(Kafka) records-lag-max`
  - `(Kinesis) millisBehindLatest`

Key Metrics

Latency

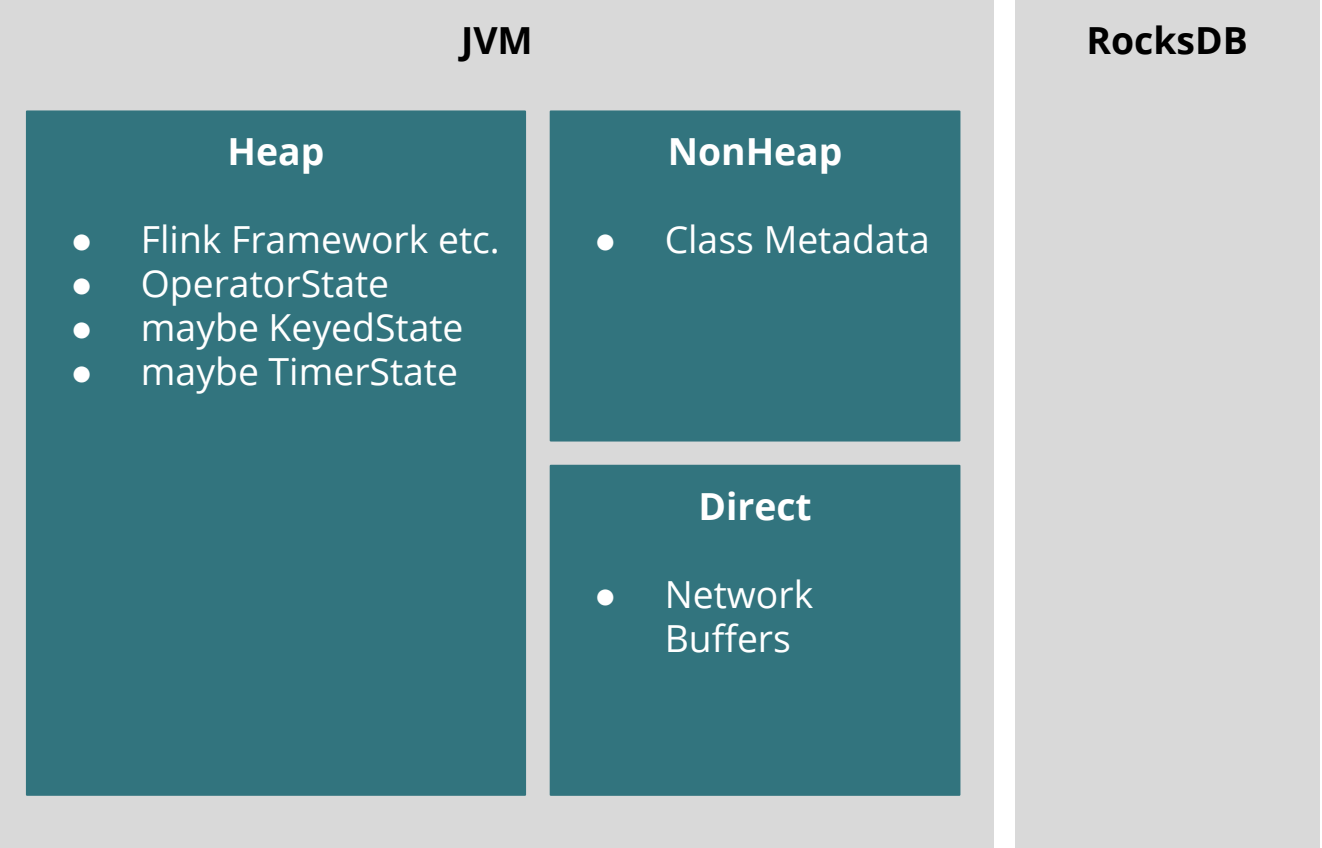
- Add timestamp to events at multiple stages, e.g.
  - event creation
  - ingestion
  - publishing
- custom metrics for reporting
  - these timestamps, or
  - the deviation from an ideal timestamp

Key Metrics for Troubleshooting

JVM Metrics

Memory

- `Status.JVM.Memory.`
  - `NonHeap.Committed`
  - `Heap.Used`
  - `Heap.Committed`
  - `Direct.MemoryUsed`
  - `Mapped.MemoryUsed`
  - `G1 Young Generation.Time`
  - `G1 Old Generation.Time`



 |

JVM Metrics

CPU

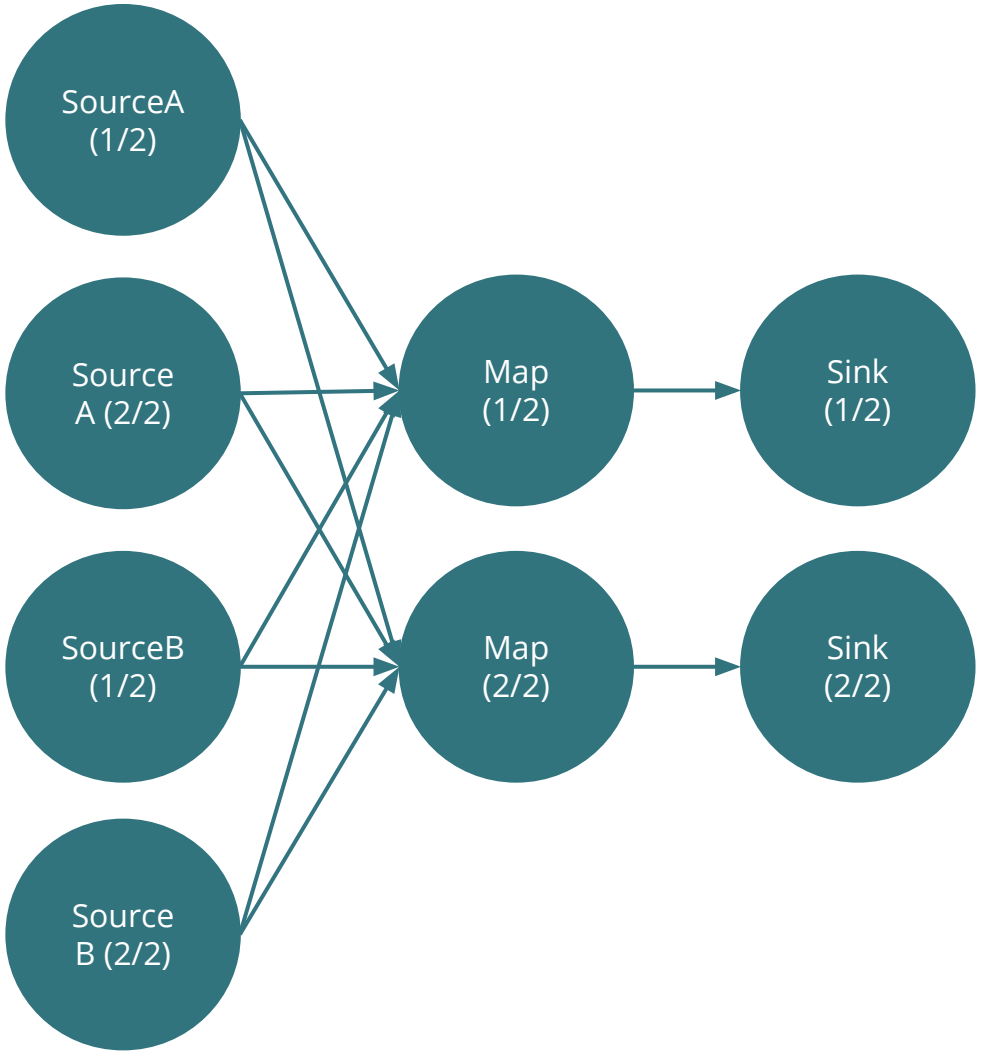
- Metrics
  - `Status.JVM.CPU.Load`
  - `Status.JVM.CPU.Time`
- Leave some slack for catch-up scenarios (& RocksDB)

Note: 0.021 = 100% load for a Taskmanager container with 1 CPU on a 48 core machine.

Troubleshooting Latency

Latency Tracking

- For each operator-subtask a latency histogram is exposed
- Enabled via `metrics.latency.interval`
- Scoped to job
- `latency.source_id.<source_id>.operator_id.<operator_id>.operator_subtask_index.<subtask_index>`



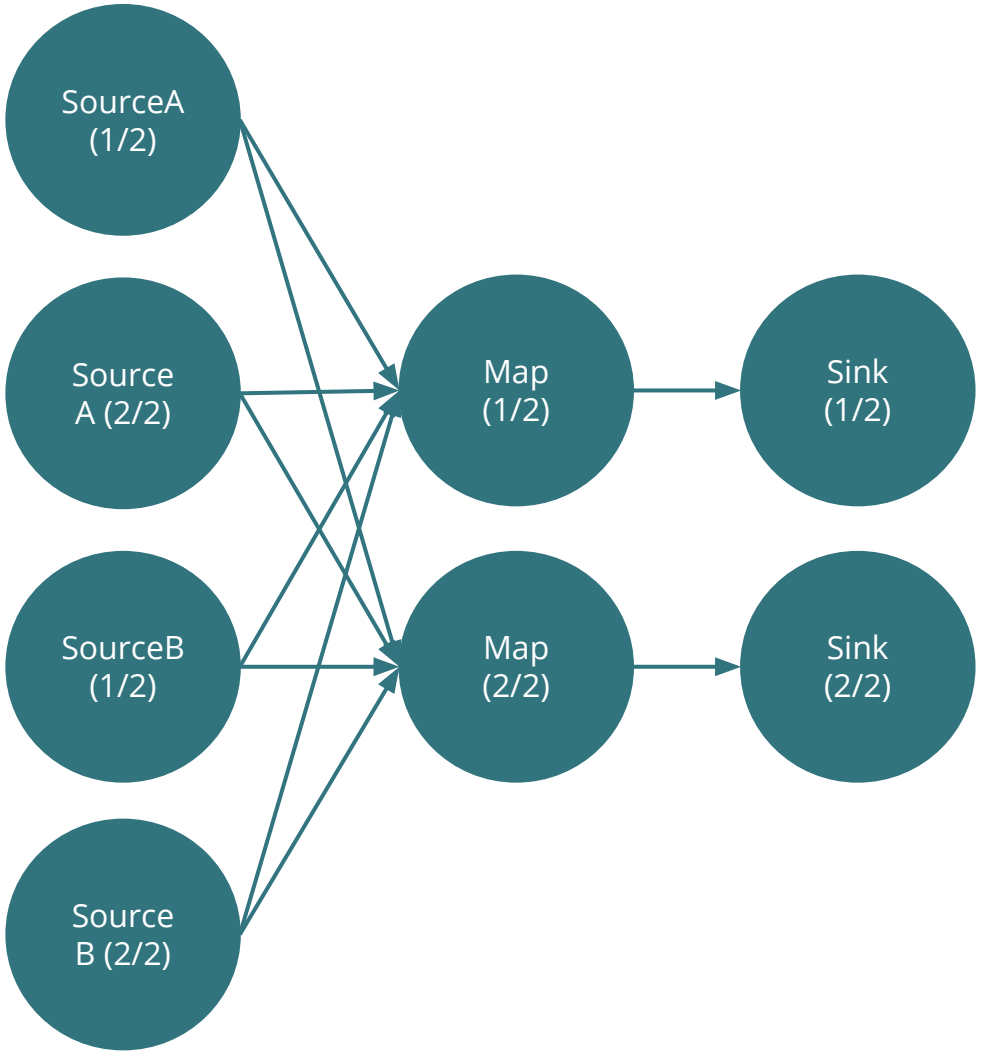
 |

Troubleshooting Latency

Latency Tracking

`metrics.latency.granularity: single`

- Per Subtask
  - Latency histogram for both sources
- Overall
  - 4 (P*#Operators)



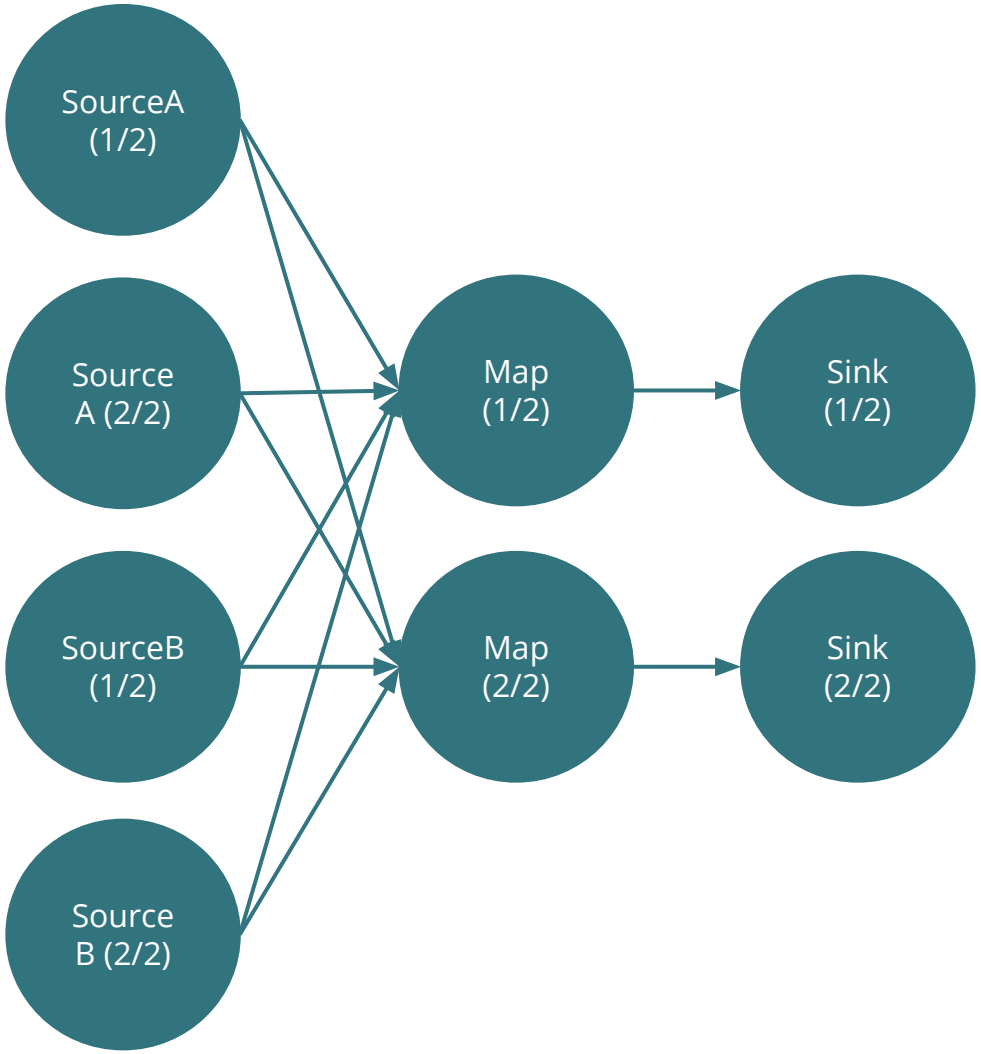
 |

Troubleshooting Latency

Latency Tracking

`metrics.latency.granularity: operator`

- Per Subtask
  - Latency histogram for Source A
  - Latency histogram for Source B
- Overall
  - 8 histograms (P * #Sources * #Operators)



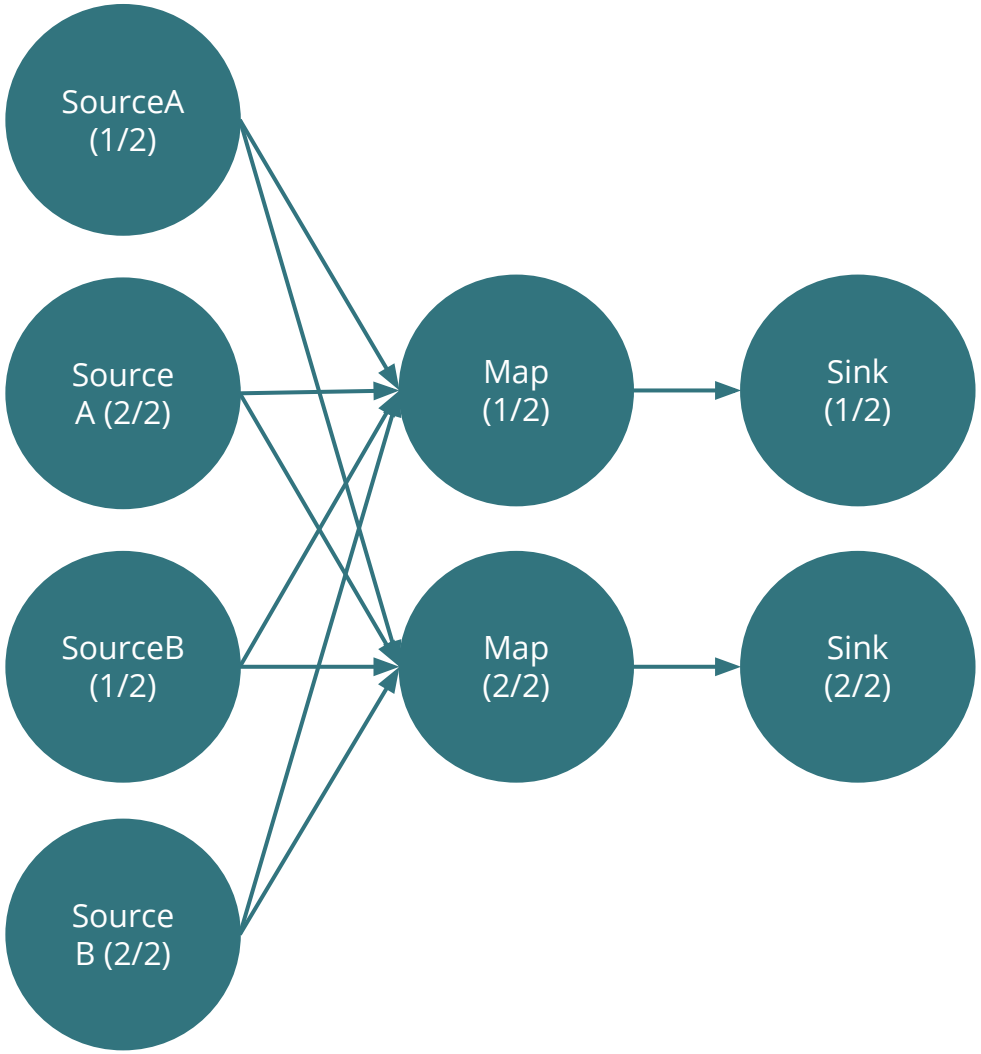
 |

Troubleshooting Latency

Latency Tracking

`metrics.latency.granularity: subtask`

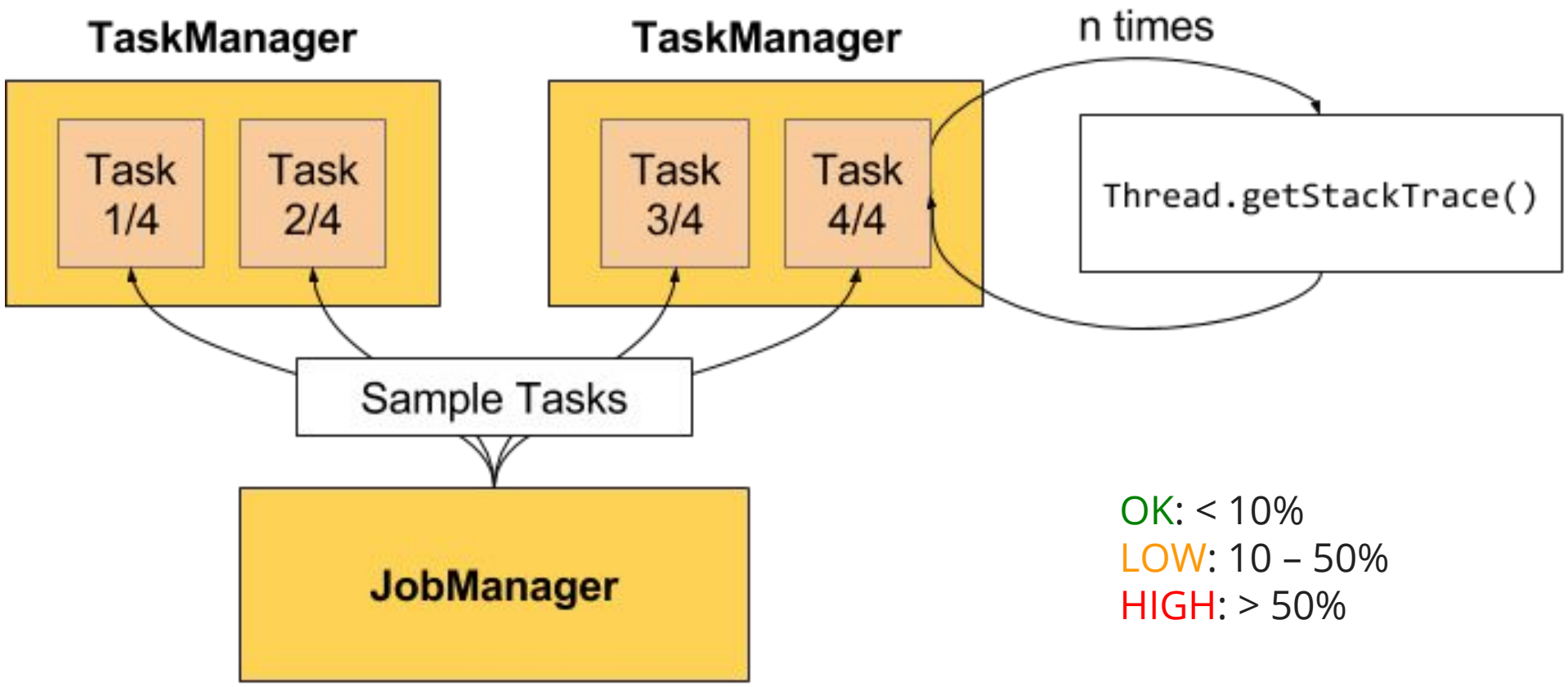
- Per Subtask
  - Latency histogram for Source A (1/2)
  - Latency histogram for Source A (2/2)
  - Latency histogram for Source B (1/2)
  - Latency histogram for Source B (2/2)
- Overall
  - 16 histogram (**P^2** * #Sources * #Operators)



 |

Troubleshooting Backpressure

Backpressure Monitor



OK: < 10%
LOW: 10 – 50%
HIGH: > 50%

Detail | SubTasks | TaskManagers | Watermarks | Accumulators | BackPressure | Metrics

Measurement: Sampling in progress... | Back Pressure Status: -

SubTask	Ratio	Status

No Data

Name	Status	Bytes Received	Records Received	Bytes Sent	Records Sent	Parallelism	Tasks
Sink: Print to Std. Out	RUNNING	184 MB	184	0 B	0	8	8
Flat Map	RUNNING	184 MB	184	184 MB	184	8	8
Map	RUNNING	184 MB	184	184 MB	184	8	8
Source: Custom Source	RUNNING	0 B	0	184 MB	184	4	4

Detail | SubTasks | TaskManagers | Watermarks | Accumulators | BackPressure | Metrics

Measurement: 17s ago | Back Pressure Status: OK

SubTask	Ratio	Status
1	0.01	OK
2	0	OK
3	0	OK
>	0	OK
5	0	OK
6	0.01	OK
7	0	OK
8	0	OK

Name	Status	Bytes Received	Records Received	Bytes Sent	Records Sent	Parallelism	Tasks
Sink: Print to Std. Out	RUNNING	1.71 GB	1,748	0 B	0	8	8
Flat Map	RUNNING	1.71 GB	1,748	1.71 GB	1,748	8	8
Map	RUNNING	1.71 GB	1,748	1.71 GB	1,748	8	8
Source: Custom Source	RUNNING	0 B	0	1.71 GB	1,748	4	4

Detail | SubTasks | TaskManagers | Watermarks | Accumulators | BackPressure | Metrics

Measurement: 1m 8s ago | Back Pressure Status: HIGH

SubTask	Ratio	Status
1	1	HIGH
2	1	HIGH
3	1	HIGH
>	1	HIGH
5	0.97	HIGH
6	1	HIGH
7	1	HIGH
8	1	HIGH

Name	Status	Bytes Received	Records Received	Bytes Sent	Records Sent	Tasks
Sink: Print to Std. Out	RUNNING	0 B	0	0 B	0	8
Flat Map	RUNNING	2.73 GB	2,792	0 B	0	8
Map	RUNNING	2.75 GB	2,800	2.73 GB	2,800	8
Source: Custom Source	RUNNING	0 B	0	2.75 GB	2,820	4

Troubleshooting Backpressure

Metrics

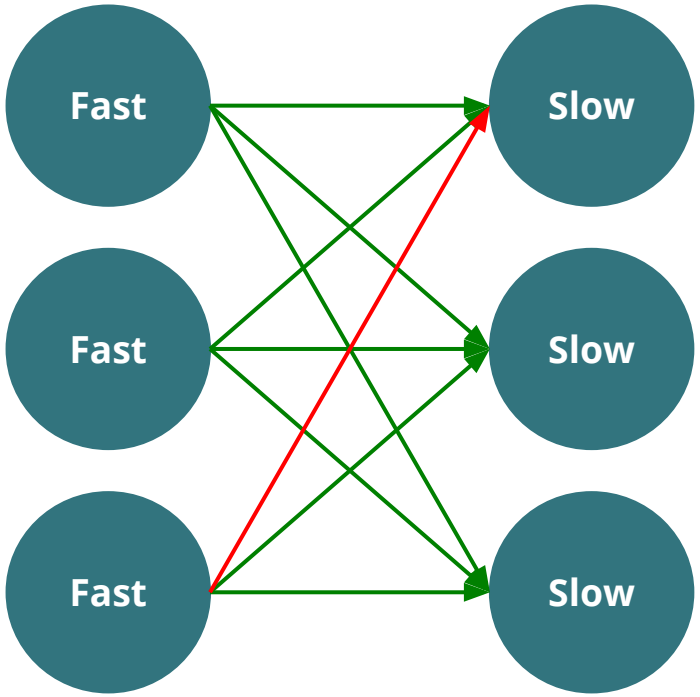
- sender's `outPoolUsage` vs. receiver's `inputFloatingBuffersUsage`
- monitored consistently
- may be used to identify the cause of backpressure
  - identify source operator
  - network latencies (credit-based flow control)
  - asymmetric backpressure
  - other resource bottleneck

https://flink.apache.org/2019/07/23/flink-network-stack-2.html

Troubleshooting Backpressure

Asymmetric Backpressure

- situation where backpressure only occurs in one channel
- hard to detect, but can lead to checkpoint timeouts
- Metrics
  - `inputFloatingBuffersUsage`, `inputExclusiveBuffersUsage`
  - `outPoolUsage`



https://flink.apache.org/2019/07/23/flink-network-stack-2.html

ververica

nico@ververica.com www.ververica.com @VervericaData