Data Stream Processing and Analytics **Prof. Vasiliki Kalavri**
Spring Semester 2020 vkalavri@bu.edu

Assignment 3

Submission deadline: March 23~~16~~ 2020, 11:59 pm

1. Introduction

In this assignment, you will use the cluster trace analytics applications you have developed to study the performance of state backends in Apache Flink. In particular, you will learn how to enable Flink's performance metrics and how to measure the latency of streaming applications.

The deliverables of this assignment consist of your **code** and a **short report** (maximum 2 pages). In your report, you will need to describe your implementation choices and trade-offs and explain the performance results you will obtain. Each task contains specific questions that need to be addressed in your report.

To get meaningful latency measurements, you will need to feed your applications with more data. Use the `gsutil` tool to download the entire job and task event tables:

// download all parts of the job event stream

```
>> gsutil cp -R gs://clusterdata-2011-2/job_events/part-000* /your/job_path/
```

// download all parts of the task event stream

```
>> gsutil cp -R gs://clusterdata-2011-2/task_events/part-000* /your/task_path/
```

Note that you will need to have at least 2GB of available space on your local disk.

2. Tasks

The assignment consists of the following two tasks. Each task's contribution to the grade is indicated inside parentheses.

a. Using Flink's state API (40/100)

1. **Task (a.1): (20/40)** Use Flink's state management API and state primitives to rewrite the `JobSchedulingLatency`, `BusyMachines`, `MaxTaskCompletionTimeFromKafka`, and `LongestSessionPerJob` exercises of the previous assignments. Make sure that any state operators maintain is implemented using a `ValueState`, `MapState`, `ListState`, `ReducingState`, or `AggregatingState`. Beware of state scoping and keep in mind that keyed state is always scoped to the key of the record currently being processed.

2. **Task (a.2): (20/40)** In the report, explain your choice of state primitives for each stateful operator. Was there an alternative approach, e.g., could you use a `ValueState` instead of `ReducingState` or a `MapState` instead of a ~~`Value`~~`ValueState<HashMap>`? What are the advantages and disadvantages of using each primitive for each particular operator?

b. Measuring latency (60/100)

In this task, you will run the stateful versions of the `JobSchedulingLatency`, `BusyMachines`, `LongestSessionPerJob` and `PerMachineStatistics` exercises and measure their performance with different state backends. Note that the window operator already uses managed state, so you don't need to make any changes to the application code for `PerMachineStatistics`.

Add a no-op sink

To avoid measuring the time of printing results to the output, add a no-op sink operator to each of your applications. You can do that by replacing the `printOrTest()` command in the end of your program with the following lines (where `T` is the type of your result stream):

```
DataStream<T> resultsStream = …

resultsStream.addSink(new SinkFunction<T>() {
  @Override
  public void invoke(T value, Context context) throws Exception {
    // no-op sink
  }
});

//printOrTest(resultStream);
```

Enable latency tracking

To enable latency tracking in Flink, you will need to add the following line in the beginning of each of your applications (after the definition of the execution environment):

```
// set latency tracking every 500ms
env.getConfig().setLatencyTrackingInterval(500);
```

This command instructs Flink to push a dummy record (the latency probe) through the dataflow every 500ms and measure how long it takes for this record to arrive all the way to the sinks. Measuring this time over a long enough period can give us a good approximation of the application's end-to-end latency.

Enable Flink metrics

In order to instruct Flink to gather metrics about your applications, you will need to enable metrics collection and execute your applications in cluster mode (not inside IntelliJ). Follow the steps below:

- Download and extract Apache Flink 1.9 (if you haven't done so already)
- Copy the `opt/flink-metrics-slf4j-*.jar` file inside the `lib/` folder.
- Add the following lines in `conf/flink-conf.yaml` to report metrics every 2s:
    a. `metrics.reporter.slf4j.class: org.apache.flink.metrics.slf4j.Slf4jReporter`
    b. `metrics.reporter.slf4j.interval: 2 SECONDS`

If you now start Flink from the command line (`./bin/start-cluster`) and submit a job, Flink will periodically report metrics and write them to the task executor's log file inside the `log/` folder. You can find relevant latency measurements under the "Histograms" section inside the log. The output will contain multiple measurements like the one below:

```
-- Histograms ------------------------------------------------------------------
10.0.0.37.taskmanager.70db0bb7ff4668af2c4f73bfc4491d3a.Job Scheduling
Latency.latency.source_id.bc764cd8ddf7a0cff126f51c16239658.operator_id.0a448493b4782967b150582570326227.operator_subtask_index.0.latency: count=8, min=1, max=9.1, mean=6.5, stddev=0.7071067811865476, p50=1.5, p75=2.0, p95=3.0, p98=4.6, p99=8.1, p999=9.1
...
```

The field "count" refers to the number of measurements collected in this particular record, and the p* values are the corresponding percentiles. For instance, p99=8.1 indicates that 99% of the collected measurements had a latency value that was below 8.1ms.

Experiments

For each of the four (4) stateful applications, perform the following experiments:

- Set the backend configuration to '**filesystem**' and run each application with **parallelism 1**, **2**, and **4**. For all executions, set the speedup factor of the job and the task events stream to 60000.
- Set the state backend to '**rocksdb**' and repeat the above experiments.

This is a total of **24 experiments** (4 applications * 2 backends * 3 parallelism settings). It is highly advised that you run each experiment at least three (3) times to reduce noise in your measurements. For each experiment, extract the relevant latency data from the corresponding log file.

For each experiment:

1. **Task (b.1) (30/60):** Report the min, max, p50, and p99 values in a table. Alternatively, you can also choose to plot a latency CDF or histogram. You can use any plotting library you like, such as gnuplot, ggplot2, matplotlib, etc.
2. **Task (b.2) (30/60):** Briefly describe and explain your results in your report. Does the latency decrease when you increase the parallelism? Why (not)? Is there an overhead when using RocksDB as the state backend as compared to using in-memory state? What could be the reason (if yes/no)?

Hint 1: Remember that you need to stop and restart your Flink cluster when you change a configuration parameter in `conf/flink-conf.yaml`.

Hint 2: Provide the parallelism as an input parameter at submission time to avoid re-building your jar every time you need to run an experiment. Run `./bin/flink -h` to see how to specify the parallelism as a runtime option.

3. Deliverables

This assignment has two (2) deliverables:

1. Your code for **Task (a.1)**. As in previous assignments, **upload your `src` folder to Blackboard**. If you have edited the `pom.xml` file in any way, make sure to provide the updated file as well. Note that **well-documented code is always easier to understand and grade**, so please make sure your code is clean, readable, and has comments.
2. A brief report (maximum 2 pages) that includes:
    a. Your answers to the questions of **Task (a.2)**
    b. The requested latency values (or plots) as described in **Task (b.1)**
    c. Your answers to the questions of **Task (b.2)**