Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Systems@ETH Zürich

Spring Term 2019 Dr. Vasiliki Kalavri

Data Stream Processing and Analytics

Assignment 1

In this exercise session, you will learn:
- How to setup and run Apache Flink on your laptop/desktop
- How to setup a development environment for Flink so you can develop streaming applications using the DataStream API
- How to package, submit, and monitor a Flink application through the Web UI

You can develop and execute Flink applications on Linux, macOS, and Windows. However, UNIX-based setups have complete tooling support and are generally preferred by Flink developers. The rest of this assignment assumes a UNIX-based setup. If you are a Windows user, you are advised to use Windows subsystem for Linux (WSL), Cygwin, or a Linux virtual machine to run Flink in a UNIX environment.

1. Setup up Apache Flink and run an example

Setup

To setup and run Flink, you need to have a working Java 8.x installation. To develop Flink applications and use its DataStream API in Java or Scala you will need a Java JDK. A Java JRE is not sufficient!

To check the correct installation of Java, open a terminal and run `java -version`.
If you have Java 8, the output will look something like this:

```
java version "1.8.0_111"
Java(TM) SE Runtime Environment (build 1.8.0_111-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.111-b14, mixed mode)
```

Download the v. 1.7.1 Flink binary from the downloads page: https://flink.apache.org/downloads.html

```
wget http://mirror.easyname.ch/apache/flink/flink-1.7.1/flink-1.7.1-
```

`bin-scala_2.11.tgz`

Go to the directory where you downloaded Flink and unpack the archive:

```
tar xzf flink-*.tgz
```

Enter the Flink home directory and start a local cluster:

```
cd flink-1.7.1
./bin/start-cluster.sh
```

You can now open your browser and visit Flink's Web UI http://localhost:8081. If everything has worked correctly, you will see one TaskManager with one avalaible slot.

Run the example

The Flink distribution you downloaded comes with runnable examples. The one we are going to use here is `SocketWindowWordCount`, an application that reads text from a socket, counts the occurrences of different words, and outputs the word counts every 5 seconds. If you want to check out the source code for this example, go to https://github.com/apache/flink/blob/release-1.7/flink-examples/flink-examples-streaming/src/main/java/org/apache/flink/streaming/examples/socket/SocketWindowWordCount.java.

Open another terminal and use `netcat` to start a local server:

```
nc -l 9000
```

In your previous terminal, submit the Flink program with the following command:

```
./bin/flink run examples/streaming/SocketWindowWordCount.jar --port
9000
```

Check the Web UI to see that the job is running. If all looks good, you can start introducing text in `nc` which will be sent to the Flink application line-by-line once you hit Enter. The program outputs its results to a log file, you can monitor with the following command:

```
tail -f log/flink-*-taskexecutor-*.out
```

Stop the Flink cluster:

```
./bin/stop-cluster.sh
```

Tasks

#1: So far, you have run Flink with a single process and parallelism of 1. To change this default setting, open the `conf/flink-conf.yaml` file and change the `taskmanager.numberOfTaskSlots` and `parallelism.default` properties. You might also want to increase the memory allocated to

you `TaskManager`, by setting the `taskmanager.heap.mb` property (in KB).

Setup a local cluster with 1 TaskManager and 4 slots (or more, depending on your laptop's hardware) and re-run the previous example with parallelism 4. Note that you will need to re-start your Flink cluster for the configuration changes to take effect.

#2: Above, you used the run action of Flink's command-line client to submit the example job. Run `./bin/flink` to see a list of available actions and options. Can you run the wordcount example with parallelism=2 without changing the configuration?

2. Setup a development environment

Software requirements

- Apache Maven 3.x. Flink provides Maven archetypes to bootstrap new projects.
- An IDE for Java and/or Scala development. Common choices are IntelliJ IDEA, Eclipse, or Netbeans with appropriate plugins installed. We recommend using IntelliJ IDEA. You can follow the instructions in https://www.jetbrains.com/idea/download/ to download and install it.

Even though Flink is a distributed data processing system, you will typically develop and run initial tests on your local machine. This makes development easier and simplifies cluster deployment, as you can run the exact same code in a cluster environment without making any changes.

To begin with, we will create a skeleton project to work from. Open a terminal, switch to a fresh working directory (ideally *not* within the Flink distribution) and run the following command:

```
mvn archetype:generate \
    -DarchetypeGroupId=org.apache.flink \
    -DarchetypeArtifactId=flink-quickstart-java \
    -DarchetypeVersion=1.7.1 \
    -DgroupId=wiki-edits \
    -DartifactId=wiki-edits \
    -Dversion=0.1 \
    -Dpackage=wikiedits \
    -DinteractiveMode=false
```

You can edit the groupId, artifactId and package if you like. With the above parameters, Maven will create a folder named `wiki-edits` and a project structure that looks like this:

```
$ tree wiki-edits
wiki-edits/
├── pom.xml
└── src
    └── main
        ├── java
```

```
        │   └── wikiedits
        │       ├── BatchJob.java
        │       ├── StreamingJob.java
        └── resources
            └── log4j.properties
```

Ingest and analyze a stream of Wikipedia Edits

As a last step we need to add the Flink Wikipedia connector as a dependency so that we can use it in our program. Edit the dependencies section of the `pom.xml` so that it looks like this:

```
<dependencies>
    <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-java</artifactId>
        <version>1.7.1</version>
    </dependency>
    <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-streaming-java_2.11</artifactId>
        <version>1.7.1</version>
    </dependency>
    <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-clients_2.11</artifactId>
        <version>1.7.1</version>
    </dependency>
    <dependency>
        <groupId>org.apache.flink</groupId>
        <artifactId>flink-connector-wikiedits_2.11</artifactId>
        <version>1.7.1</version>
    </dependency>
</dependencies>
```

Open your IDE and import the Maven project (Import Project -> Select pom.xml). Using the `src/main/java/wikiedits/StreamingJob.java` as a template, create a new file `src/main/java/wikiedits/WikipediaAnalysis.java`.

The template already contains the definition of a `StreamExecutionEnvironment`. We can now add a source that ingests events corresponding to wikipedia edits as follows:

```
DataStream<WikipediaEditEvent> edits = env.addSource(new
WikipediaEditsSource());
```

The WikipediaEditsSource connects to the **#en.wikipedia IRC** channel and reads edit events. The following code groups ingested events by user ID and continuously aggregates the byte changes made by each user:

```
DataStream<Tuple2<String, Integer>> result = edits
    // project the event user and the diff
     .map(new MapFunction<WikipediaEditEvent, Tuple2<String,
Integer>>() {
        @Override
        public Tuple2<String, Integer> map(WikipediaEditEvent event) {
                return new Tuple2<>(
                    event.getUser(), event.getByteDiff());
        }
     })
     // group by user
     .keyBy(0)
     // aggregate changes per user
     .reduce(new ReduceFunction<Tuple2<String, Integer>>() {
        @Override
        public Tuple2<String, Integer> reduce(
            Tuple2<String, Integer> e1, Tuple2<String, Integer> e2) {
                return new Tuple2<>(e1.f0, e1.f1 + e2.f1);
        }
     });
```

(Remark: this can be reformulated more elegantly using lambda expressions, but you will need to add explicit type annotations: https://ci.apache.org/projects/flink/flink-docs-stable/dev/java_lambdas.html)

You can now print the result to standard output by adding the following command:

```
result.print();
```

If you now run your `WikipediaAnalysis` program, you should see output like this:

```
4> (Rangasyd,-15)
6> (Seokgjin,8)
8> (MONGO,-18)
5> (4.129.194.107,65)
8> (Narky Blert,61)
3> (AntiCedros,567)
6> (Ecuador593,105)
```

Tasks

#1: Add a filter transformation after the aggregation to drop negative byte changes. Check the Flink documentation for an example.

#2: The above program maintains and continuously prints byte diffs for all users. If many distinct users are active, this might lead to severe memory usage as all counts for all users need to be tracked at all

times. Change the program to aggregate byte diffs every 10s instead, by using a `timeWindow()` method. See the Flink documentation for an example.