CprE 419 Lab 10: Stream Data Processing Using Flink

Department of Electrical and Computer Engineering
Iowa State University

Purpose

In this lab, the goal is to learn how to use *streaming windows* in Apache Flink to solve some interesting problems in streaming fashion.

During this lab, you will write programs with pipelined jobs to:

- Using Flink platform on top of YARN
- Different types of *streaming windows* in streaming data processing.
- Some real world problems that can be solved using Flink

Submission

Create a single zip archive with the following and hand it in through Canvas**:**

- Screenshots of your results for each individual experiment.
- Commented Code for your program. Include all source files needed for compilation and make sure it compiles successfully. Make sure you output the results to a specified folder of each experiment.

Flink resources

Apache Flink documentation:
https://ci.apache.org/projects/flink/flink-docs-stable/
Running a Flink job on YARN:
https://ci.apache.org/projects/flink/flink-docs-release-1.4/ops/deployment/yarn_setup.html
Batch Flink sample problems:
https://ci.apache.org/projects/flink/flink-docs-release-1.4/dev/batch/examples.html
Streaming Flink sample problems:
https://github.com/apache/flink/tree/master/flink-examples

Flink Installation and execution

Download the Apache Flink package from the following link. Note, we recommend to download the version 1.4.2 since the experiments of this lab are tested on Apache Flink 1.4.2.
http://flink.apache.org/downloads.html

After having downloaded the package, unpack it and move it to your home directory (or any sub-directory of your choice) in the cluster (hpc-class.its.iastate.edu). You can download the flink package in your local machine, then copy it to the cluser, using the following command
scp flink-1.4.2 user_name@hpc-class.its.iastate.edu:/home/user_name/

Otherwise, you can download it directly from the cluster, using following commands:

- Download the Flink:
```
wget http://mirrors.advancedhosters.com/apache/flink/flink-1.4.2/flink-1.4.2-bin-hadoop26-scala_2.11.tgz
```

- Unpack the package:
```
tar -xvzf flink-1.4.2-bin-hadoop26-scala_2.11.tgz
```

- Remove the package (after unpacked it):
```
rm flink-1.4.2-bin-hadoop26-scala_2.11.tgz
```

- Add Flink to the PATH environment from the namenode(hpc-class-hdp01) (optional):
```
export PATH=$PATH:/home/<Net_ID>/flink-1.4.2/bin
```

To test Flink, run the following command:
flink run flink-1.4.2/examples/batch/WordCount.jar

In order to be able to compile a Flink program in Eclipse, you need to have the following dependencies in your maven project:

```
<!-- https://mvnrepository.com/artifact/org.apache.flink/flink-java -->
<dependency>
	<groupId>org.apache.flink</groupId>
	<artifactId>flink-java</artifactId>
	<version>1.4.2</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.flink/flink-core -->
<dependency>
	<groupId>org.apache.flink</groupId>
	<artifactId>flink-core</artifactId>
	<version>1.4.2</version>
</dependency>

<!-- https://mvnrepository.com/artifact/org.apache.flink/flink-streaming-java -->
<dependency>
	<groupId>org.apache.flink</groupId>
	<artifactId>flink-streaming-java_2.11</artifactId>
	<version>1.4.2</version>
</dependency>
```

Make sure you have JVM (>=1.8.x), Maven(>=3.1.x). Sample Stream Word Count implementation with its pom.xml configurations is provided on Canvas. It is run in a streaming fashion and reads the file from HDFS. Run the following command:
flink run –m yarn-cluster –yarncontainer 1 JAR_NAME.jar

In order to check your output result, you may see it in the logs by running the following command: **yarn logs –applicationId YOUR_APPLICATION_ID**

Input Stream

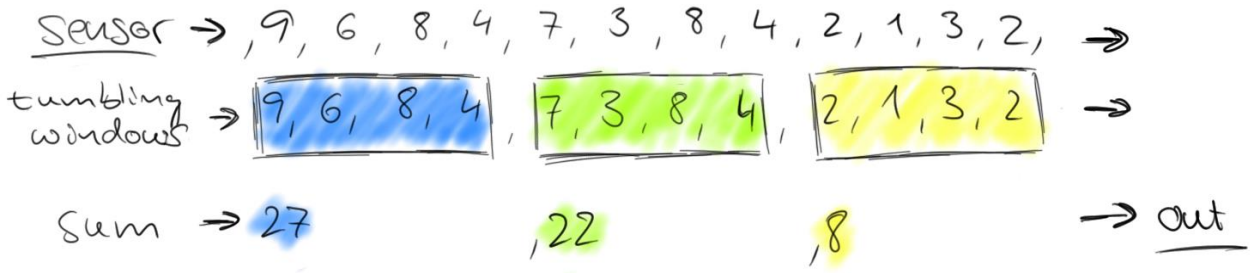
Given a stream of records. Each record is a string, containing information about an open source project, attributes are separated by comma, as follow:

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	repository	language	architecture	community	continuous_i	document	history	license	managem	size	unit_test	state	stars
2	matplotlib/matplotlib.github.com	Python	0.770463	2	0	0.014931	2.297872	0	0.212766	1575488	0.013242	active	5
3	NCIP/c3pr-docs	Java	0.997449	3	0	0.087444	1.434211	0	0	765164	0	dormant	0
4	AnXgotta/Sur	C++	0.714286	1	0	0.123698	0	0	0	2155	0	dormant	0
5	bigloupe/SoS-JobScheduler	Java	0.957573	3	1	0.315557	11.42857	1	0	657960	0.007257	None	1
6	barons/zf_shop	Ruby	0.381323	3	0	0.327179	0	1	0	472610	0.055335	None	0
7	uzleo/hiwi	C++	0.865123	2	0	0.218128	15.8	1	0	170144	0.011772	None	0
8	berlinonline/banned_books	PHP	0.44	4	0	0.017882	5	1	0	399320	0	None	0
9	pszabolcs/canvasandroid	Java	0.988235	4	0	0.136708	32.66667	0	0	119414	0	None	0
10	mk12/mycraft	Java	0.662614	1	0	0.326084	3.583333	1	0	134913	0.117074	None	7
11	BulldogDrummond/etmod	C	0.820513	1	0	0.085501	0	0	0	220996	0.00994	None	0
12	ryseto/stodyn	C++	0.943548	1	0	0.14322	0	1	0	228026	0	None	0
13	UfSoft/ILog	Python	0.666667	1	0	0.186233	14.16667	0	0	6004	0	None	0
14	nix858/osu	C++	1	1	0	0.18365	0	0	0	3648	0	None	0
15	WilbertHo/foobar	Python	0.705882	2	0	0.320261	4.666667	0	0	210	0.251282	None	0
16	kaludis/epoll-echo-server	C	1	1	0	0.36478	0	0	0	303	0.090592	None	0
17	Jarcionek/MTG-Deck-Builder	Java	0.983051	1	0	0.014853	0	0	0	3051	0.514811	None	0

Using Fink, you can simulate the stream by reading line-by-line from hdfs file at: /cpre419/github.csv.

Experiment 1 (50 points) Tumbling windows

Tumbling windows discretize a stream into non-overlapping windows. We can compute the windows based on time, or based on number of records. Then you can apply your aggregate function in each window. For example: in the following stream, each 4 elements become a window, then we output the sum for each window.



In the given Github stream, for each window of 200,000 records, report the number of repositories and name of repository that has the highest stars number for each language, as following.

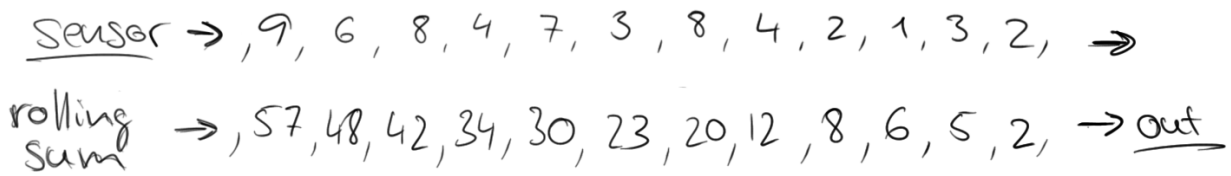
First window:
<language_1> <num_of_repo> <name_of_repo_highest_start> <num_starts>
<language_2> <num_of_repo> <name_of_repo_highest_start> <num_starts>
….

Second window:
<language_1> <num_of_repo> <name_of_repo_highest_start> <num_starts>
<language_2> <num_of_repo> <name_of_repo_highest_start> <num_starts>
….

For each window, sort the results in descending order of num_of_repo.

Experiment 2 (50 points) Infinite Windows

You can keep the size of your processing window to be infinity. In which you maintain the state of your aggregate function along the stream and update it whenever you see a new record. For example, in the following stream, where new records are adding to the left, we maintain the sum of the element we seen so far:



In the given Github stream, report the number of repositories and name of repository that has the highest starts number for each language. Sort the results in descending order of num_of_repo, and print out the result after each 200,000 records. (Due to the high volume of the stream DON'T print out your results after each record, even though you have to maintain it along the stream and update for each new record).