Identifying Similar Documents in a Stream

Department of Electrical and Computer Engineering
Iowa State University
Spring 2013

Purpose

The goal of this lab is to write software that can in real-time, analyze a stream of documents, to find out near-duplicates in this stream.

In writing this application using Infosphere Streams, you will need to write a new primitive operator in C++ (or Java), and use this within the main application, written in SPL. Using C++ or Java provides additional flexibility in programming, when compared with directly using SPL. Writing your own operator also allows for code reuse.

Submission

Create a zip archive with the following and hand it in through blackboard.

- A data-flow diagram for the SPL application
- Commented Code for your program. Include all source files needed for compilation.

Experiment

> Consider a continuous stream of news articles streaming in from multiple blogs and news sites. These updates come in the form of a link and each link leads to one article. It is common to have multiple similar, and sometimes near-identical news documents, coming from different sources, all close to each other in time. It will be good if such duplicates are filtered out and only one representative document is presented to the user for consumption, from each set of nearly similar documents.

You are given as input a stream of file names, each name leading to a single document, stored on the filesystem. For convenience, this stream is also stored in a file that you can read using a FileSource operator.

At the top of each hour, you are required to aggregate all the documents received within the hour, cluster them into groups of near-identical documents, and return a single document from each cluster (you can return an arbitrary document from each cluster). Since you can track time only when an event occurs, generate an output for each hour on the occurrence of the first event at the top of next hour. For identifying near-duplicates, you can use the shingling technique that we also used in a previous lab.

Since the code for aggregation can be hard to write using direct SPL, we would like you to write your own primitive operator, using C++ (or Java).

Inputs:
The input given to you is a file “Filenames.csv” which contains the name of several files, saved in “/datasets/Lab10” folder. The tuples in “Filenames.csv” is of the format
<timestamp>,<filename>
<filename> gives absolute pathnames. For instance, if there is a document named “file-1.txt” stored in “/datasets/Lab10/Documents”, the corresponding tuple in “Filenames.csv” is :
<timestamp>,file-1.txt

Several files are stored in the folder “/datasets/Lab10/Documents”, each corresponding to one filename in “Filenames.csv”.

Outputs:
The output must be produced every hour and must contain a stream comprising of an arbitrary file from each cluster of identical documents. The output tuple must be of the format:
<timestamp>,<filename>

where <timestamp> is be the time when the file <filename> appears in the stream, and has the following format: hh:mm:ss.sss.

Primitive Operators

A primitive operator is written using C++ or Java. There are two kinds of primitive operators – generic and non-generic. We will focus on the non-generic primitive operator which is almost purely written in C++ or Java.

The location where we store the user-defined operator is very important. **Namespaces** provide the means to logically group a set of related operators. Putting operators into meaningful namespaces helps maintain modularity and prevents name clashing with operators, functions, and types in other namespaces. So if you were to create a “Main.spl” file inside a directory “Example”, and “Main.spl” application uses two user-defined operators – *op1* and *op2*, then both needs to be placed in a subdirectory of “Example”, say we call it “my.ops” in a separate directory “op1” and op2. Hence, the namespace for operator “op1” is my.ops::op1 and that for “op2” is my.ops::op2. You have to use this namespace in the Main.spl operator in order to be able to use the operator.

C++ Primitive Operator

“spl-make-operator --kind c++” command is used to create a basic template for C++ primitive operators when inside the folder specified by namespace. On running this command, two code generation templates “.cgt” file is created - one for header (eg op1_h.cgt, if created for operator “op1”) and one for cpp (eg op1_cpp.cgt, if created for “op1”). The codes are written within this template.

You can find examples and more details in the following link:

http://pic.dhe.ibm.com/infocenter/streams/v2r0/index.jsp?nav=%2F2_1_4

Java Primitive Operator

“spl-make-operator --kind java” command is used to create a <operator-name>.xml file within the folder specified by the namespace, which is required for creating Java primitive operator. The XML file is edited to replace the dummy class name <className>My Operator</className>, under the <executionSettings> section with the appropriate Java Class name. Unlike C++ primitive operator, no code generation template is created for Java. After writing the Java code for the operator, say “op1.java”, you have to compile the code and include “op1.class” in the namespace folder.

After writing the code in the .cgt template (for C++ operator), or including the java class file in the namespace folder (for Java operator), compile the SPL code and run the application in the same way as you have done for the previous SPL applications.

To understand better about Java primitive operators, please read the following sections from Streams Information Center :
1) IBM Streams Processing Language Toolkit Development Reference and
2) IBM Streams Processing Language Operator Model Reference.

Note: Please make sure to add the following lines in ".bashrc" file in your home folder if you are using a Java primitive operator:

```
export JAVA_HOME=/opt/ibm/java-x86_64-60

export PATH=/opt/ibm/java-x86_64-60/bin:$PATH
```

An example of Java and C++ primitive operator are attached with this report and also uploaded in the Resources section.

Programming Hints

Primitive Operator:
Create a folder named "Lab10" which will contain the main SPL application, a subfolder named "my.op". Create a folder within "my.op" named "DeSimilarDocs" where you will generate the operator templates (for C++), an XML file <operator-nam>.xml using "spl-make-operator" command. The operator name in this case will be "DeSimilarDocs". The window size can be given as a parameter to this operator.