Getting Started

Table of contents

1. Pig Setup

1.1. Requirements

Mandatory

Unix and Windows users need the following:

- **Hadoop 2.X** - http://hadoop.apache.org/common/releases.html (You can run Pig with different versions of Hadoop by setting HADOOP_HOME to point to the directory where you have installed Hadoop. If you do not set HADOOP_HOME, by default Pig will run with the embedded version, currently Hadoop 2.7.3.)
- **Java 1.7** - http://java.sun.com/javase/downloads/index.jsp (set JAVA_HOME to the root of your Java installation)

Optional

- **Python 2.7** - https://www.python.org (when using Streaming Python UDFs)
- **Ant 1.8** - http://ant.apache.org/ (for builds)

1.2. Download Pig

To get a Pig distribution, do the following:

1. Download a recent stable release from one of the Apache Download Mirrors (see Pig Releases).
2. Unpack the downloaded Pig distribution, and then note the following:
   - The Pig script file, pig, is located in the bin directory (/pig-n.n.n/bin/pig). The Pig environment variables are described in the Pig script file.
   - The Pig properties file, pig.properties, is located in the conf directory (/pig-n.n.n/conf/pig.properties). You can specify an alternate location using the PIG_CONF_DIR environment variable.
3. Add /pig-n.n.n/bin to your path. Use export (bash,sh,ksh) or setenv (tcsh,csh). For example:
   ```
   $ export PATH=/<my-path-to-pig>/pig-n.n.n/bin:$PATH
   ```
4. Test the Pig installation with this simple command: `$ pig -help`

1.3. Build Pig

To build pig, do the following:

1. Check out the Pig code from SVN: `svn co`

`http://svn.apache.org/repos/asf/pig/trunk`

2. Build the code from the top directory: `ant`
   If the build is successful, you should see the pig.jar file created in that directory.
3. Validate the pig.jar by running a unit test: `ant test`

2. Running Pig

You can run Pig (execute Pig Latin statements and Pig commands) using various modes.

	Local Mode	**Tez Local Mode**	**Spark Local Mode**	**Mapreduce Mode**	**Tez Mode**	**Spark Mode**
Interactive Mode	yes	experimental	yes	yes		
Batch Mode	yes	experimental	yes	yes		

2.1. Execution Modes

Pig has six execution modes or exectypes:

- **Local Mode** - To run Pig in local mode, you need access to a single machine; all files are installed and run using your local host and file system. Specify local mode using the -x flag (pig -x local).
- **Tez Local Mode** - To run Pig in tez local mode. It is similar to local mode, except internally Pig will invoke tez runtime engine. Specify Tez local mode using the -x flag (pig -x tez_local).

  Note: Tez local mode is experimental. There are some queries which just error out on bigger data in local mode.
- **Spark Local Mode** - To run Pig in spark local mode. It is similar to local mode, except internally Pig will invoke spark runtime engine. Specify Spark local mode using the -x flag (pig -x spark_local).

  Note: Spark local mode is experimental. There are some queries which just error out on bigger data in local mode.
- **Mapreduce Mode** - To run Pig in mapreduce mode, you need access to a Hadoop cluster and HDFS installation. Mapreduce mode is the default mode; you can, *but don't need to*, specify it using the -x flag (pig OR pig -x mapreduce).
- **Tez Mode** - To run Pig in Tez mode, you need access to a Hadoop cluster and HDFS installation. Specify Tez mode using the -x flag (-x tez).
- **Spark Mode** - To run Pig in Spark mode, you need access to a Spark, Yarn or Mesos cluster and HDFS installation. Specify Spark mode using the -x flag (-x spark). In Spark

execution mode, it is necessary to set env::SPARK_MASTER to an appropriate value (local - local mode, yarn-client - yarn-client mode, mesos://host:port - spark on mesos or spark://host:port - spark cluster. For more information refer to spark documentation on Master URLs, *yarn-cluster mode is currently not supported*). Pig scripts run on Spark can take advantage of the dynamic allocation feature. The feature can be enabled by simply enabling *spark.dynamicAllocation.enabled*. Refer to spark configuration for additional configuration details. In general all properties in the pig script prefixed with *spark.* are copied to the Spark Application Configuration. Please note that Yarn auxillary service need to be enabled on Spark for this to work. See Spark documentation for additional details.

You can run Pig in either mode using the "pig" command (the bin/pig Perl script) or the "java" command (java -cp pig.jar ...).

2.1.1. Examples

This example shows how to run Pig in local and mapreduce mode using the pig command.

```
/* local mode */
$ pig -x local ...

/* Tez local mode */
$ pig -x tez_local ...

/* Spark local mode */
$ pig -x spark_local ...

/* mapreduce mode */
$ pig ...
or
$ pig -x mapreduce ...

/* Tez mode */
$ pig -x tez ...

/* Spark mode */
$ pig -x spark ...
```

2.2. Interactive Mode

You can run Pig in interactive mode using the Grunt shell. Invoke the Grunt shell using the "pig" command (as shown below) and then enter your Pig Latin statements and Pig commands interactively at the command line.

2.2.1. Example

These Pig Latin statements extract all user IDs from the /etc/passwd file. First, copy the /etc/passwd file to your local working directory. Next, invoke the Grunt shell by typing the "pig" command (in local or hadoop mode). Then, enter the Pig Latin statements interactively at the grunt prompt (be sure to include the semicolon after each statement). The DUMP operator will display the results to your terminal screen.

```
grunt> A = load 'passwd' using PigStorage(':');
grunt> B = foreach A generate $0 as id;
grunt> dump B;
```

Local Mode

```
$ pig -x local
... - Connecting to ...
grunt>
```

Tez Local Mode

```
$ pig -x tez_local
... - Connecting to ...
grunt>
```

Spark Local Mode

```
$ pig -x spark_local
... - Connecting to ...
grunt>
```

Mapreduce Mode

```
$ pig -x mapreduce
... - Connecting to ...
grunt>

or

$ pig
... - Connecting to ...
grunt>
```

Tez Mode

```
$ pig -x tez
... - Connecting to ...
grunt>
```

Spark Mode

```
$ pig -x spark
... - Connecting to ...
grunt>
```

2.3. Batch Mode

You can run Pig in batch mode using Pig scripts and the "pig" command (in local or hadoop mode).

2.3.1. Example

The Pig Latin statements in the Pig script (id.pig) extract all user IDs from the /etc/passwd file. First, copy the /etc/passwd file to your local working directory. Next, run the Pig script from the command line (using local or mapreduce mode). The STORE operator will write the results to a file (id.out).

```
/* id.pig */

A = load 'passwd' using PigStorage(':');  -- load the passwd file
B = foreach A generate $0 as id;  -- extract the user IDs
store B into 'id.out';  -- write the results to a file name id.out
```

Local Mode

```
$ pig -x local id.pig
```

Tez Local Mode

```
$ pig -x tez_local id.pig
```

Spark Local Mode

```
$ pig -x spark_local id.pig
```

Mapreduce Mode

```
$ pig id.pig
or
$ pig -x mapreduce id.pig
```

Tez Mode

```
$ pig -x tez id.pig
```

Spark Mode

```
$ pig -x spark id.pig
```

2.3.2. Pig Scripts

Use Pig scripts to place Pig Latin statements and Pig commands in a single file. While not required, it is good practice to identify the file using the *.pig extension.

You can run Pig scripts from the command line and from the Grunt shell (see the run and exec commands).

Pig scripts allow you to pass values to parameters using parameter substitution.

Comments in Scripts

You can include comments in Pig scripts:

- For multi-line comments use /* …. */
- For single-line comments use --

```
/* myscript.pig
My script is simple.
It includes three Pig Latin statements.
*/

A = LOAD 'student' USING PigStorage() AS (name:chararray, age:int,
gpa:float); -- loading data
B = FOREACH A GENERATE name;  -- transforming data
DUMP B;  -- retrieving results
```

Scripts and Distributed File Systems

Pig supports running scripts (and Jar files) that are stored in HDFS, Amazon S3, and other distributed file systems. The script's full location URI is required (see REGISTER for information about Jar files). For example, to run a Pig script on HDFS, do the following:

```
$ pig hdfs://nn.mydomain.com:9020/myscripts/script.pig
```

3. Running jobs on a Kerberos secured cluster

Kerberos is a authentication system that uses tickets with a limited validity time.
As a consequence running a pig script on a kerberos secured hadoop cluster limits the running time to at most the remaining validity time of these kerberos tickets. When doing really complex analytics this may become a problem as the job may need to run for a longer time than these ticket times allow.

3.1. Short lived jobs

When running short jobs all you need to do is ensure that the user has been logged in into Kerberos via the normal kinit method.
The Hadoop job will automatically pickup these credentials and the job will run fine.

3.2. Long lived jobs

A kerberos keytab file is essentially a Kerberos specific form of the password of a user.
It is possible to enable a Hadoop job to request new tickets when they expire by creating a keytab file and make it part of the job that is running in the cluster. This will extend the maximum job duration beyond the maximum renew time of the kerberos tickets.

Usage:

1. Create a keytab file for the required principal.
   Using the ktutil tool you can create a keytab using roughly these commands:
   ```
   addent -password -p niels@EXAMPLE.NL -k 1 -e rc4-hmac
   addent -password -p niels@EXAMPLE.NL -k 1 -e aes256-cts
   wkt niels.keytab
   ```
2. Set the following properties (either via the .pigrc file or on the command line via -P file)
   - *java.security.krb5.conf*
     The path to the local krb5.conf file.
     Usually this is "/etc/krb5.conf"
   - *hadoop.security.krb5.principal*
     The pricipal you want to login with.
     Usually this would look like this "niels@EXAMPLE.NL"
   - *hadoop.security.krb5.keytab*
     The path to the local keytab file that must be used to authenticate with.
     Usually this would look like this "/home/niels/.krb/niels.keytab"

NOTE: All paths in these variables are local to the client system starting the actual pig script. This can be run without any special access to the cluster nodes.

Overall you would create a file that looks like this (assume we call it niels.kerberos.properties):

```
java.security.krb5.conf=/etc/krb5.conf
hadoop.security.krb5.principal=niels@EXAMPLE.NL
hadoop.security.krb5.keytab=/home/niels/.krb/niels.keytab
```

and start your script like this:

```
pig -P niels.kerberos.properties script.pig
```

4. Pig Latin Statements

Pig Latin statements are the basic constructs you use to process data using Pig. A Pig Latin statement is an operator that takes a relation as input and produces another relation as output. (This definition applies to all Pig Latin operators except LOAD and STORE which read data from and write data to the file system.) Pig Latin statements may include expressions and schemas. Pig Latin statements can span multiple lines and must end with a semi-colon (;). By default, Pig Latin statements are processed using multi-query execution.

Pig Latin statements are generally organized as follows:

- A LOAD statement to read data from the file system.
- A series of "transformation" statements to process the data.
- A DUMP statement to view results or a STORE statement to save the results.

Note that a DUMP or STORE statement is required to generate output.

- In this example Pig will validate, but not execute, the LOAD and FOREACH statements.

```
A = LOAD 'student' USING PigStorage() AS (name:chararray, age:int,
gpa:float);
B = FOREACH A GENERATE name;
```

- In this example, Pig will validate and then execute the LOAD, FOREACH, and DUMP statements.

```
A = LOAD 'student' USING PigStorage() AS (name:chararray, age:int,
gpa:float);
B = FOREACH A GENERATE name;
DUMP B;
(John)
(Mary)
(Bill)
(Joe)
```

4.1. Loading Data

Use the LOAD operator and the load/store functions to read data into Pig (PigStorage is the default load function).

4.2. Working with Data

Pig allows you to transform data in many ways. As a starting point, become familiar with these operators:

- Use the FILTER operator to work with tuples or rows of data. Use the FOREACH operator to work with columns of data.
- Use the GROUP operator to group data in a single relation. Use the COGROUP, inner JOIN, and outer JOIN operators to group or join data in two or more relations.
- Use the UNION operator to merge the contents of two or more relations. Use the SPLIT operator to partition the contents of a relation into multiple relations.

4.3. Storing Intermediate Results

Pig stores the intermediate data generated between MapReduce jobs in a temporary location on HDFS. This location must already exist on HDFS prior to use. This location can be configured using the pig.temp.dir property. The property's default value is "/tmp" which is the same as the hardcoded location in Pig 0.7.0 and earlier versions.

4.4. Storing Final Results

Use the STORE operator and the load/store functions to write results to the file system (PigStorage is the default store function).

Note: During the testing/debugging phase of your implementation, you can use DUMP to display results to your terminal screen. However, in a production environment you always want to use the STORE operator to save your results (see Store vs. Dump).

4.5. Debugging Pig Latin

Pig Latin provides operators that can help you debug your Pig Latin statements:

- Use the DUMP operator to display results to your terminal screen.
- Use the DESCRIBE operator to review the schema of a relation.
- Use the EXPLAIN operator to view the logical, physical, or map reduce execution plans to compute a relation.
- Use the ILLUSTRATE operator to view the step-by-step execution of a series of statements.

Shortcuts for Debugging Operators

Pig provides shortcuts for the frequently used debugging operators (DUMP, DESCRIBE, EXPLAIN, ILLUSTRATE). These shortcuts can be used in Grunt shell or within pig scripts. Following are the shortcuts supported by pig

- \d alias - shourtcut for DUMP operator. If alias is ignored last defined alias will be used.
- \de alias - shourtcut for DESCRIBE operator. If alias is ignored last defined alias will be used.
- \e alias - shourtcut for EXPLAIN operator. If alias is ignored last defined alias will be used.
- \i alias - shourtcut for ILLUSTRATE operator. If alias is ignored last defined alias will be used.

- \q - To quit grunt shell

5. Pig Properties

Pig supports a number of Java properties that you can use to customize Pig behavior. You can retrieve a list of the properties using the help properties command. All of these properties are optional; none are required.

To specify Pig properties use one of these mechanisms:

- The pig.properties file (add the directory that contains the pig.properties file to the classpath)
- The -D and a Pig property in PIG_OPTS environment variable (export PIG_OPTS=-Dpig.tmpfilecompression=true)
- The -P command line option and a properties file (pig -P mypig.properties)
- The set command (set pig.exec.nocombiner true)

Note: The properties file uses standard Java property file format.

The following precedence order is supported: pig.properties < -D Pig property < -P properties file < set command. This means that if the same property is provided using the –D command line option as well as the –P command line option (properties file), the value of the property in the properties file will take precedence.

To specify Hadoop properties you can use the same mechanisms:

- Hadoop configuration files (include pig-cluster-hadoop-site.xml)
- The -D and a Hadoop property in PIG_OPTS environment variable (export PIG_OPTS=–Dmapreduce.task.profile=true)
- The -P command line option and a property file (pig -P property_file)
- The set command (set mapred.map.tasks.speculative.execution false)

The same precedence holds: Hadoop configuration files < -D Hadoop property < -P properties_file < set command.

Hadoop properties are not interpreted by Pig but are passed directly to Hadoop. Any Hadoop property can be passed this way.

All properties that Pig collects, including Hadoop properties, are available to any UDF via the UDFContext object. To get access to the properties, you can call the getJobConf method.

6. Pig Tutorial

The Pig tutorial shows you how to run Pig scripts using Pig's local mode, mapreduce mode, Tez mode and Spark mode (see Execution Modes).

To get started, do the following preliminary tasks:

1. Make sure the JAVA_HOME environment variable is set the root of your Java installation.
2. Make sure your PATH includes bin/pig (this enables you to run the tutorials using the "pig" command).
   ```
   $ export PATH=/<my-path-to-pig>/pig-0.17.0/bin:$PATH
   ```
3. Set the PIG_HOME environment variable:
   ```
   $ export PIG_HOME=/<my-path-to-pig>/pig-0.17.0
   ```
4. Create the pigtutorial.tar.gz file:
   - Move to the Pig tutorial directory (.../pig-0.17.0/tutorial).
   - Run the "ant" command from the tutorial directory. This will create the pigtutorial.tar.gz file.
5. Copy the pigtutorial.tar.gz file from the Pig tutorial directory to your local directory.
6. Unzip the pigtutorial.tar.gz file.
   ```
   $ tar -xzf pigtutorial.tar.gz
   ```
7. A new directory named pigtmp is created. This directory contains the Pig Tutorial Files. These files work with Hadoop 0.20.2 and include everything you need to run Pig Script 1 and Pig Script 2.

6.1. Running the Pig Scripts in Local Mode

To run the Pig scripts in local mode, do the following:

1. Move to the pigtmp directory.
2. Execute the following command (using either script1-local.pig or script2-local.pig).
   ```
   $ pig -x local script1-local.pig
   ```
   Or if you are using Tez local mode:
   ```
   $ pig -x tez_local script1-local.pig
   ```
   Or if you are using Spark local mode:
   ```
   $ pig -x spark_local script1-local.pig
   ```
3. Review the result files, located in the script1-local-results.txt directory. The output may contain a few Hadoop warnings which can be ignored:
   ```
   2010-04-08 12:55:33,642 [main] INFO
   org.apache.hadoop.metrics.jvm.JvmMetrics
   ```

```
- Cannot initialize JVM Metrics with processName=JobTracker, sessionId=
- already initialized
```

6.2. Running the Pig Scripts in Mapreduce Mode, Tez Mode or Spark Mode

To run the Pig scripts in mapreduce mode, do the following:

1. Move to the pigtmp directory.
2. Copy the excite.log.bz2 file from the pigtmp directory to the HDFS directory.

   ```
   $ hadoop fs –copyFromLocal excite.log.bz2 .
   ```
3. Set the PIG_CLASSPATH environment variable to the location of the cluster configuration directory (the directory that contains the core-site.xml, hdfs-site.xml and mapred-site.xml files):

   ```
   export PIG_CLASSPATH=/mycluster/conf
   ```

   If you are using Tez, you will also need to put Tez configuration directory (the directory that contains the tez-site.xml):

   ```
   export PIG_CLASSPATH=/mycluster/conf:/tez/conf
   ```

   If you are using Spark, you will also need to specify SPARK_HOME and specify SPARK_JAR which is the hdfs location where you uploaded $SPARK_HOME/lib/spark-assembly*.jar:

   ```
   export SPARK_HOME=/mysparkhome/; export
   SPARK_JAR=hdfs://example.com:8020/spark-assembly*.jar
   ```

   Note: The PIG_CLASSPATH can also be used to add any other 3rd party dependencies or resource files a pig script may require. If there is also a need to make the added entries take the highest precedence in the Pig JVM's classpath order, one may also set the env-var PIG_USER_CLASSPATH_FIRST to any value, such as 'true' (and unset the env-var to disable).
4. Set the HADOOP_CONF_DIR environment variable to the location of the cluster configuration directory:

   ```
   export HADOOP_CONF_DIR=/mycluster/conf
   ```
5. Execute the following command (using either script1-hadoop.pig or script2-hadoop.pig):

   ```
   $ pig script1-hadoop.pig
   ```

   Or if you are using Tez:

   ```
   $ pig -x tez script1-hadoop.pig
   ```

   Or if you are using Spark:

   ```
   $ pig -x spark script1-hadoop.pig
   ```
6. Review the result files, located in the script1-hadoop-results or script2-hadoop-results

HDFS directory:

```
$ hadoop fs -ls script1-hadoop-results
$ hadoop fs -cat 'script1-hadoop-results/*' | less
```

6.3. Pig Tutorial Files

The contents of the Pig tutorial file (pigtutorial.tar.gz) are described here.

File	Description
pig.jar	Pig JAR file
tutorial.jar	User defined functions (UDFs) and Java classes
script1-local.pig	Pig Script 1, Query Phrase Popularity (local mode)
script1-hadoop.pig	Pig Script 1, Query Phrase Popularity (mapreduce mode)
script2-local.pig	Pig Script 2, Temporal Query Phrase Popularity (local mode)
script2-hadoop.pig	Pig Script 2, Temporal Query Phrase Popularity (mapreduce mode)
excite-small.log	Log file, Excite search engine (local mode)
excite.log.bz2	Log file, Excite search engine (mapreduce)

The user defined functions (UDFs) are described here.

UDF	Description
ExtractHour	Extracts the hour from the record.
NGramGenerator	Composes n-grams from the set of words.
NonURLDetector	Removes the record if the query field is empty or a URL.
ScoreGenerator	Calculates a "popularity" score for the n-gram.

ToLower	Changes the query field to lowercase.
TutorialUtil	Divides the query string into a set of words.

6.4. Pig Script 1: Query Phrase Popularity

The Query Phrase Popularity script (script1-local.pig or script1-hadoop.pig) processes a search query log file from the Excite search engine and finds search phrases that occur with particular high frequency during certain times of the day.

The script is shown here:

- Register the tutorial JAR file so that the included UDFs can be called in the script.

```
REGISTER ./tutorial.jar;
```

- Use the PigStorage function to load the excite log file (excite.log or excite-small.log) into the "raw" bag as an array of records with the fields **user**, **time**, and **query**.

```
raw = LOAD 'excite.log' USING PigStorage('\t') AS (user, time, query);
```

- Call the NonURLDetector UDF to remove records if the query field is empty or a URL.

```
clean1 = FILTER raw BY org.apache.pig.tutorial.NonURLDetector(query);
```

- Call the ToLower UDF to change the query field to lowercase.

```
clean2 = FOREACH clean1 GENERATE user, time,
org.apache.pig.tutorial.ToLower(query) as query;
```

- Because the log file only contains queries for a single day, we are only interested in the hour. The excite query log timestamp format is YYMMDDHHMMSS. Call the ExtractHour UDF to extract the hour (HH) from the time field.

```
houred = FOREACH clean2 GENERATE user,
org.apache.pig.tutorial.ExtractHour(time) as hour, query;
```

- Call the NGramGenerator UDF to compose the n-grams of the query.

```
ngramed1 = FOREACH houred GENERATE user, hour,
flatten(org.apache.pig.tutorial.NGramGenerator(query)) as ngram;
```

- Use the DISTINCT operator to get the unique n-grams for all records.

```
ngramed2 = DISTINCT ngramed1;
```

- Use the GROUP operator to group records by n-gram and hour.

```
hour_frequency1 = GROUP ngramed2 BY (ngram, hour);
```

- Use the COUNT function to get the count (occurrences) of each n-gram.

```
hour_frequency2 = FOREACH hour_frequency1 GENERATE flatten($0), COUNT($1)
as count;
```

- Use the GROUP operator to group records by n-gram only. Each group now corresponds to a distinct n-gram and has the count for each hour.

```
uniq_frequency1 = GROUP hour_frequency2 BY group::ngram;
```

- For each group, identify the hour in which this n-gram is used with a particularly high frequency. Call the ScoreGenerator UDF to calculate a "popularity" score for the n-gram.

```
uniq_frequency2 = FOREACH uniq_frequency1 GENERATE flatten($0),
flatten(org.apache.pig.tutorial.ScoreGenerator($1));
```

- Use the FOREACH-GENERATE operator to assign names to the fields.

```
uniq_frequency3 = FOREACH uniq_frequency2 GENERATE $1 as hour, $0 as ngram,
$2 as score, $3 as count, $4 as mean;
```

- Use the FILTER operator to remove all records with a score less than or equal to 2.0.

```
filtered_uniq_frequency = FILTER uniq_frequency3 BY score > 2.0;
```

- Use the ORDER operator to sort the remaining records by hour and score.

```
ordered_uniq_frequency = ORDER filtered_uniq_frequency BY hour, score;
```

- Use the PigStorage function to store the results. The output file contains a list of n-grams with the following fields: **hour**, **ngram**, **score**, **count**, **mean**.

```
STORE ordered_uniq_frequency INTO '/tmp/tutorial-results' USING
PigStorage();
```

6.5. Pig Script 2: Temporal Query Phrase Popularity

The Temporal Query Phrase Popularity script (script2-local.pig or script2-hadoop.pig) processes a search query log file from the Excite search engine and compares the occurrence of frequency of search phrases across two time periods separated by twelve hours.

The script is shown here:

- Register the tutorial JAR file so that the user defined functions (UDFs) can be called in the script.

```
REGISTER ./tutorial.jar;
```

- Use the PigStorage function to load the excite log file (excite.log or excite-small.log) into the “raw” bag as an array of records with the fields **user**, **time**, and **query**.

```
raw = LOAD 'excite.log' USING PigStorage('\t') AS (user, time, query);
```

- Call the NonURLDetector UDF to remove records if the query field is empty or a URL.

```
clean1 = FILTER raw BY org.apache.pig.tutorial.NonURLDetector(query);
```

- Call the ToLower UDF to change the query field to lowercase.

```
clean2 = FOREACH clean1 GENERATE user, time,
org.apache.pig.tutorial.ToLower(query) as query;
```

- Because the log file only contains queries for a single day, we are only interested in the hour. The excite query log timestamp format is YYMMDDHHMMSS. Call the ExtractHour UDF to extract the hour from the time field.

```
houred = FOREACH clean2 GENERATE user,
org.apache.pig.tutorial.ExtractHour(time) as hour, query;
```

- Call the NGramGenerator UDF to compose the n-grams of the query.

```
ngramed1 = FOREACH houred GENERATE user, hour,
flatten(org.apache.pig.tutorial.NGramGenerator(query)) as ngram;
```

- Use the DISTINCT operator to get the unique n-grams for all records.

```
ngramed2 = DISTINCT ngramed1;
```

- Use the GROUP operator to group the records by n-gram and hour.

```
hour_frequency1 = GROUP ngramed2 BY (ngram, hour);
```

- Use the COUNT function to get the count (occurrences) of each n-gram.

```
hour_frequency2 = FOREACH hour_frequency1 GENERATE flatten($0), COUNT($1)
as count;
```

- Use the FOREACH-GENERATE operator to assign names to the fields.

```
hour_frequency3 = FOREACH hour_frequency2 GENERATE $0 as ngram, $1 as hour,
$2 as count;
```

- Use the FILTERoperator to get the n-grams for hour ‘00’

```
hour00 = FILTER hour_frequency2 BY hour eq '00';
```

- Uses the FILTER operators to get the n-grams for hour ‘12’

```
hour12 = FILTER hour_frequency3 BY hour eq '12';
```

- Use the JOIN operator to get the n-grams that appear in both hours.

```
same = JOIN hour00 BY $0, hour12 BY $0;
```

- Use the FOREACH-GENERATE operator to record their frequency.

```
same1 = FOREACH same GENERATE hour_frequency2::hour00::group::ngram as
ngram, $2 as count00, $5 as count12;
```

- Use the PigStorage function to store the results. The output file contains a list of n-grams with the following fields: **ngram**, **count00**, **count12**.

```
STORE same1 INTO '/tmp/tutorial-join-results' USING PigStorage();
```