Map-Reduce based Word Count

Demo Application Guide

Version 1.0

March-2014

[Introduction 3](#_Toc382318111)

[Inputting Data Files 3](#_Toc382318112)

[Input File Formats 3](#_Toc382318113)

[Input Splits 3](#_Toc382318114)

[Record Reader 4](#_Toc382318115)

[Mapper 4](#_Toc382318116)

[Combiner 4](#_Toc382318117)

[Reducer 4](#_Toc382318118)

[Partition and Shuffle 4](#_Toc382318119)

[HDFS Adapter 4](#_Toc382318120)

[Pre-Requisites 5](#_Toc382318121)

[Hadoop Services 5](#_Toc382318122)

[dtgateway Service 5](#_Toc382318123)

[Assumptions 6](#_Toc382318124)

[DataTorrent Install Location 6](#_Toc382318125)

[Location of Packaged Demo Applications 6](#_Toc382318126)

[Launching the Demo Applications (demos.jar) using dtcli 6](#_Toc382318127)

[Terms Used 6](#_Toc382318128)

[Map-Reduce Word Count Application 7](#_Toc382318129)

[Map-Reduce Word Count Application Functionality 7](#_Toc382318130)

[Application Directed Acyclic Graph (DAG) 7](#_Toc382318131)

[Application components 7](#_Toc382318132)

[Operators 8](#_Toc382318133)

[Streams 8](#_Toc382318134)

[Application Source Code 8](#_Toc382318135)

[Configuration 8](#_Toc382318136)

[HDFS Input Directory 8](#_Toc382318137)

[HDFS Output File 9](#_Toc382318138)

[Number of Mapper Jobs 9](#_Toc382318139)

[Number of Reducer Jobs 9](#_Toc382318140)

[Updating ~/.dt/dt-site.xml file 9](#_Toc382318141)

[How to Launch Word Count application (dtcli) 10](#_Toc382318142)

[How to monitor the Application 10](#_Toc382318143)

[Monitoring the application using dtcli 10](#_Toc382318144)

[Monitoring the application using DataTorrent Gateway Console 11](#_Toc382318145)

[How to view Word Count Results 11](#_Toc382318146)

[Using Hadoop NameNode UI 11](#_Toc382318147)

[Using HDFS commands 15](#_Toc382318148)

[Glossary of Terms used 17](#_Toc382318149)

# Introduction

Map-Reduce Word Count application demonstrates one of the very important platform features which is to allow users to reuse their Map-Reduce code.

Typically, still lot of enterprises that have made a Hadoop infra-structure investments are using solutions that are built using Map-Reduce Jobs. Real-time stream processing paradigm is a different application paradigm than the typical Map-Reduce paradigm. While, Hadoop provides a distributed infra-structure to run Map-Reduce based applications over data files stored in HDFS.

The Word Count applications generates the unique word counts in the given documents. Thus, if the word "real-time" appears twice in documents Foo1 and Foo2 then the line,

{Real-time=2}

will appear in the output. Similarly if the word "streaming" appears four times in documents Foo1 and Foo2, then the line:

{streaming=4}

will appear in the output.

The Word Count application, uses a Map-Reduce job to produce counts of unique words. In order to execute the Map-Reduce job it provides wrapper operators to handle Map and Reduce functions. The application simulates Hadoop’s Map-Reduce paradigm and fits it into DataTorrent Platform application.

This demonstrates how Map-Reduce features are supported on DataTorrent platform and how such features can be utilized in stream processing applications running on DataTorrent Platform.

## Inputting Data Files

DataTorrent platform can receive input from various sources. One of the input sources can be input files from HDFS. This application demonstrates how to read files from HDFS, which is the mechanism used for reading in the input data by Hadoop Map-Reduce applications. DataTorrent platform has open-sourced various input adapters that allows applications to received inputs from variety of input sources. These input sources can be Kafka, RabbitMQ or even a network socket. Similarly, HDFS input adapter can be one of the input sources from which DataTorrent platform can receive inputs.

## Input File Formats

Hadoop Map-Reduce supports different types of file formats (TextInputFormat, KeyValueInputFormat, SequenceFileInputFormat) for the input files that are read from HDFS. DataTorrent platform reuses the APIs created by Hadoop for reading the data with mentioned file formats. No additional coding is necessary.

## Input Splits

As input data is passed to the Mapper jobs in Hadoop Map-Reduce, the data read from the HDFS files, gets split into chunks of data so that the only part of the data can be passed to each Mapper job for processing. This application demonstrates the similar capability by the Partitioning support provided by DataTorrent platform. The application reuses the APIs created by Hadoop with some additional coding so as to create data partitions as required by DataTorrent Platform.

## Record Reader

The application reuses the APIs created by Hadoop to create the records from the input splits as created by *Input Splits* mechanism mentioned above. The record reader provides the necessary abstraction to convert the data read from file into the Key-Value pairs as expected by the Mapper job.

## Mapper

The application reuses the Mapper job code created to run on Hadoop Map-Reduce. The application simulates the Hadoop Map-Reduce data structures as needed by the Mapper function and facilitates the Mapper function to run on DataTorrent platform.

## Combiner

The application also allows execution of the Combiner function in the same way Hadoop Map-Reduce does. Application executes the Combiner function (which is same as Reduce function) immediately after the Mapper function is executed.

## Reducer

The application reuses the Reducer job code created to run on Hadoop Map-Reduce. The application simulates the Hadoop Map-Reduce data structures as needed by the Reducer function and facilitates the Reducer function to run on DataTorrent platform.

## Partition and Shuffle

DataTorrent platform has the ability to partition the stream of data tuples based on the given partitioning scheme. The default partitioning scheme uses the hash value of the individual keys of the data tuples to partition the key-value pairs to further operations which in this case is Reducer function. Based on the hash values of the keys, DataTorrent platform shuffles the keys with the same value to the same Reducer job.

## HDFS Adapter

This application also demonstrates DataTorrent platform ability to interact with HDFS for reading files as well as writing to it.

# Pre-Requisites

The document assumes that following Hadoop and DataTorrent services are running before using this document,

## Hadoop Services

In order to be able to launch Demo Applications on the DataTorrent platform, Hadoop 2.2 (Yarn) grid needs to be running. The Hadoop install can be from any of the distributors or from Apache (<http://hadoop.apache.org/>) as long as it is Hadoop 2.2.

## dtgateway Service

Please make sure that “dtgateway” service is running. This can be checked by executing following command on the command shell,

$ sudo service dtgateway status

# Assumptions

This document assumes following about the location where the Demo applications are placed.

## Location of Packaged Demo Applications

DataTorrent platform ships with number of Demo applications. You will find a Jar file (demos.jar) containing Demo applications under DataTorrent installation directory. The Jar file demos.jar contains multiple pre-built demo applications that user can launch on the Apache Hadoop cluster. The DataTorrent command line interface (dtcli) also present in the DataTorrent installation folder can be used to launch these Demo applications on a Hadoop cluster.

## Launching the Demo Applications (demos.jar) using dtcli

Please refer to Getting-Started-Guide to find out more about how to use “dtcli” and also how to launch the demo applications (demos.jar) that are shipped with DataTorrent platform.

# Terms Used

To find out more about the terms used this document, please refer to section [Glossary](#_Glossary_of_Terms) in this document.

# Map-Reduce Word Count Application

This section talks about following,

1. [Map-Reduce Word Count Application Functionality](#_Map-Reduce_Application_Functionalit)
2. [How to launch the application](#_How_to_Launch)
3. [How to monitor the application](#_How_to_monitor)
4. [How to view word count results](#_How_to_view)

## Map-Reduce Word Count Application Functionality

Application functionality is explained as below,

1. The Map input operator in the application reads the input file from HDFS
2. The input operator splits the log files into multiple splits. One Mapper function is associated with each split.
3. The input file record gets read by Hadoop’s Record Reader converting it into Key-Value pairs as needed by the Mapper function. The Input File Format used is Hadoop’s TextInputFormat which reads one line from the input file and returns that as Value in the Key-Value pair.
4. Mapper function then looks at each data tuple and extracts individual unique words on the line and creates counting objects with each unique word as Key along with Count “1” as value.
5. The output from Mapper objects is then passed to the Combiner function which aggregates the counts for the given unique word key.
6. The output from Combiner function is then fed to Reducer function which then aggregates the total count for each unique word key and writes it to HDFS.

### Application Directed Acyclic Graph (DAG)

Figure 1 Application DAG (TO BE ADDED)

### Application components

#### Operators

Logs Counter application has following Operators as shown in the above DAG,

|  |  |
| --- | --- |
| **Operator** | **Description** |
| map | This is an input operator that reads input from HDFS. It acts as a wrapper for the Map function and provides necessary hooks so that Map function can be executed inside DataTorrent framework. |
| reduce | This operator acts as a wrapper for the Reduce function and provides necessary hooks so that Reduce function can be executed inside DataTorrent framework |
| console | This is an output operator that writes the results to HDFS. |

#### Streams

|  |  |
| --- | --- |
| **Stream** | **Description** |
| input\_map | Stream of data tuples containing key and value-list generated by map operator. |
| input\_count\_map | Stream counts for number of mappers used |
| Console\_reduce | Stream of Counts along with respective words. |

### Application Source Code

Source code for the Operators used in the application can be viewed from links given below –

|  |  |  |
| --- | --- | --- |
| **Operator** | **Source** | **Library** |
| map | [MapOperator.java](https://github.com/DataTorrent/Malhar/blob/master/demos/src/main/java/com/datatorrent/demos/mroperator/MapOperator.java) | Custom |
|  | [AbstractHdfsInputOperator.java](https://github.com/DataTorrent/Malhar/blob/master/library/src/main/java/com/datatorrent/lib/io/fs/AbstractHDFSInputOperator.java) | Malhar Library |
| reduce | [ReduceOperator.java](https://github.com/DataTorrent/Malhar/blob/master/demos/src/main/java/com/datatorrent/demos/mroperator/ReduceOperator.java) | Custom |
| console | [HdfsKeyValOutputOperator.java](https://github.com/DataTorrent/Malhar/blob/master/demos/src/main/java/com/datatorrent/demos/mroperator/HdfsKeyValOutputOperator.java) | Custom |
|  | [AbstractHdfsOutputOperator.java](https://github.com/DataTorrent/Malhar/blob/master/library/src/main/java/com/datatorrent/lib/io/fs/AbstractHdfsOutputOperator.java) | Malhar Library |

Source code for the wrapper application class can be viewed here –

* [NewWordCountApplication.java](https://github.com/DataTorrent/Malhar/blob/master/demos/src/main/java/com/datatorrent/demos/mroperator/NewWordCountApplication.java)
* [WordCount.java](https://github.com/DataTorrent/Malhar/blob/master/demos/src/main/java/com/datatorrent/demos/mroperator/WordCount.java) (Map-Reduce Job)

## Configuration

Word Count Demo application requires following properties set through DataTorrent configuration file,

1. HDFS Input Directory
2. HDFS Output File
3. Number of Mapper Jobs
4. Number of Reducer Jobs

### HDFS Input Directory

This HDFS directory should contain the input files that will be used by Word-Count application. The input data files need to be copied under this location. Sample input files can be downloaded from [GitHub](https://github.com/DataTorrent/Malhar/tree/master/demos/src/main/resources/com/datatorrent/demos/mroperator) source location – [Input Data Files](https://github.com/DataTorrent/Malhar/tree/master/demos/src/main/resources/com/datatorrent/demos/mroperator). The source directory contains following three files that are used for Word Count Application,

1. [sample1.text](https://github.com/DataTorrent/Malhar/tree/master/demos/src/main/resources/com/datatorrent/demos/mroperator/sample1.txt)
2. [sample2.txt](https://github.com/DataTorrent/Malhar/tree/master/demos/src/main/resources/com/datatorrent/demos/mroperator/sample2.txt)
3. [sample3.txt](https://github.com/DataTorrent/Malhar/tree/master/demos/src/main/resources/com/datatorrent/demos/mroperator/sample3.txt)

Please use appropriate Hadoop HDFS commands to copy above files to the HDFS Input Directory. For example, assuming command hadoop is in the path, file sample1.txt is in the current directory and HDFS Input Directory name is wordcountin

$ hdfs dfs –copyFromLocal sample1.txt wordcountin

### HDFS Output File

Word Count Application generates the output in HDFS Output file. The name of the output file needs to be specified as shown in the next section. For this illustration let’s assume the name of HDFS output file to be “wordcountout”.

### Number of Mapper Jobs

As name indicates, this configuration defines number of mapper jobs to be run. This is only hint to the MapReduce Framework but actual number of mappers are still determined the framework.

### Number of Reducer Jobs

As name indicates, this configuration defines number of reducer jobs to be run.

### Specification

1. The configuration properties need to be specified in a configuration file. The configuration file can be specified when launching the application. If one is not specified, the platform uses the configuration file ~/.dt/dt-site.xml by default. Please open ~/.dt/dt-site.xml file in an editor. If one is not present please create a new file.
2. Add following XML snippet,

<?xml version="1.0" encoding="UTF-8"?>

<configuration>

<!—Map-Reduce Word Count demo -->

<property>

<name>com.datatorrent.demos.mroperator.NewWordCountApplication.inputDirName</name>

<value>HDFS Directory Path for Input Files</value>

</property>

<property>

<name>com.datatorrent.demos.mroperator.NewWordCountApplication.outputDirName</name>

<value>Generated Output File name</value>

</property>

<property>

<name>com.datatorrent.demos.mroperator.NewWordCountApplication.numOfMaps</name>

<value>2</value>

</property>

<property>

<name>com.datatorrent.demos.mroperator.NewWordCountApplication.numOfReducers</name>

<value>1</value>

</property>

</configuration>

## How to Launch Word Count application (dtcli)

The demo applications jar contains the application. On launching the jar from “dtcli” the list of applications packaged in the jar appears as shown in the screenshot below. The entry that is shown for each application is the path of the application class unless an alias was specified for the class in “dt-site.xml” in which case the alias is shown. The Word Count Application Id appears to be Id “4” as can be seen in below screenshot. Please specify this Id “4” on the “dtcli” ” command shell, as also can be seen in the below screenshot,

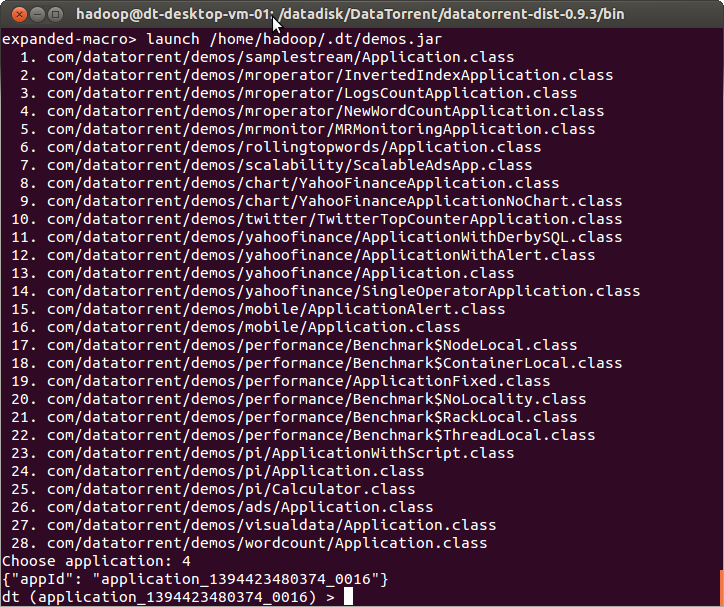


Figure Specify Application Id for Inverted Indexer on dtcli

## How to monitor the Application

Application launched through “dtcli” can be monitored through two mechanisms,

1. Using “dtcli” command shell
2. Using DataTorrent Gateway Console

### Monitoring the application using dtcli

Twitter Demo application can be monitored through the “dtcli” command shell. Following are some of the useful commands that can be used to monitor as well as control the state of the application,

1. get-app-info – Print details for the selected application, details such as – total allocated memory, number of allocated containers, total numbers of tuples processed etc.
2. show-logical-plan – Print details about the logical plan for the selected application
3. show-physical-plan - Print details about the physical plan for the selected application
4. kill-app – Kill this application

For further details about various available commands please refer to [Operations and Installation Guide](https://www.datatorrent.com/docs/guides/OperationandInstallationGuide.html).

### Monitoring the application using DataTorrent Gateway Console

Twitter Demo application can now be monitored through DataTorrent Gateway Console. The launched application gets listed in the “*Application List*” widget as shown below,

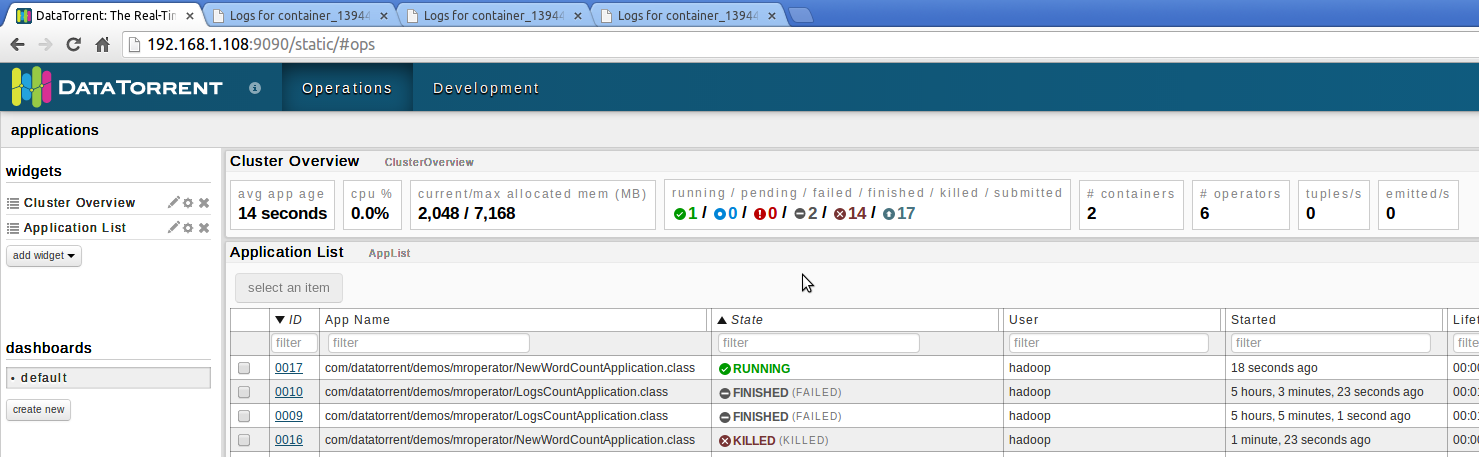


Figure Monitoring Application

On clicking the Application Id link (under the Id column), you can see the application state and further details such as -

1. Application Id and Version information
2. Application State (Running/Finished etc.)
3. Total number of Tuples Processed
4. Total number of Tuples Emitted
5. Latency (milliseconds)
6. Logical Directed Acyclic Graph (DAG)

Please refer to screenshot below that displays the above details,

Figure Demo Application Details (**TO BE ADDED**)

## How to view Word Count Results

Word Count Application generates the results into the HDFS Output File that’s mentioned in the configuration. There are couple of ways the results can be viewed,

1. Using Hadoop’s Web-based monitoring UI
2. Using HDFS commands

### Using Hadoop NameNode UI

Hadoop provides a web-based UI that can be utilized to monitor Hadoop NameNode details. With the default configuration, the NameNode front page is at http://namenode-name:50070/. It lists the DataNodes in the cluster and basic statistics of the cluster. This web interface can be used to browse HDFS (using "Browse the filesystem" link on the NameNode front page). Following screenshot displays the NameNode UI assuming the NameNode is running on host 192.168.1.108 on port 50070,

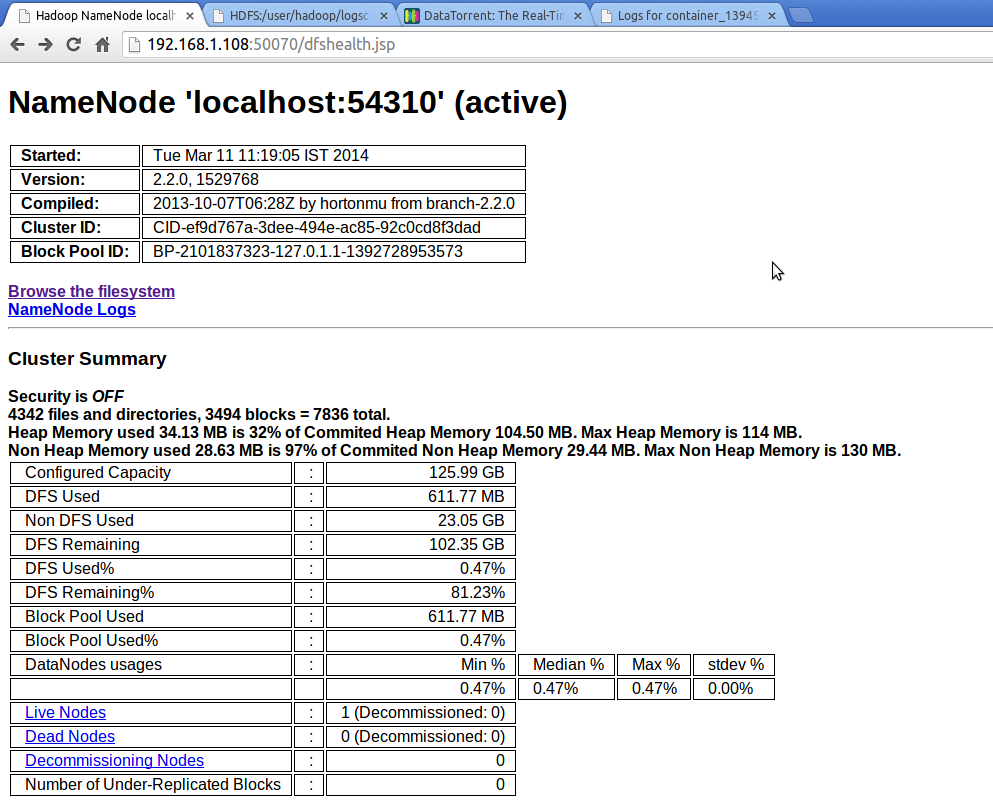


Figure Hadoop NameNode UI

Clicking the link “Browse the filesystem” will bring up following UI that displays HDFS details,

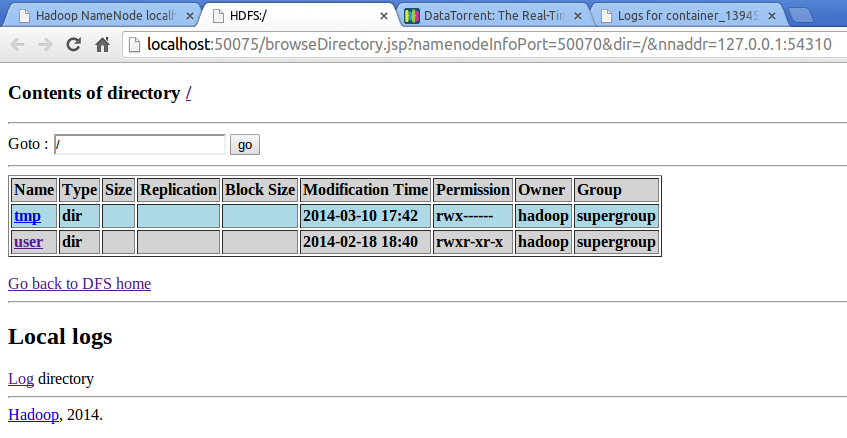


Figure Hadoop HDFS Directory "/"

On clicking, the “user” link above, following UI will get displayed for directory “/user”,

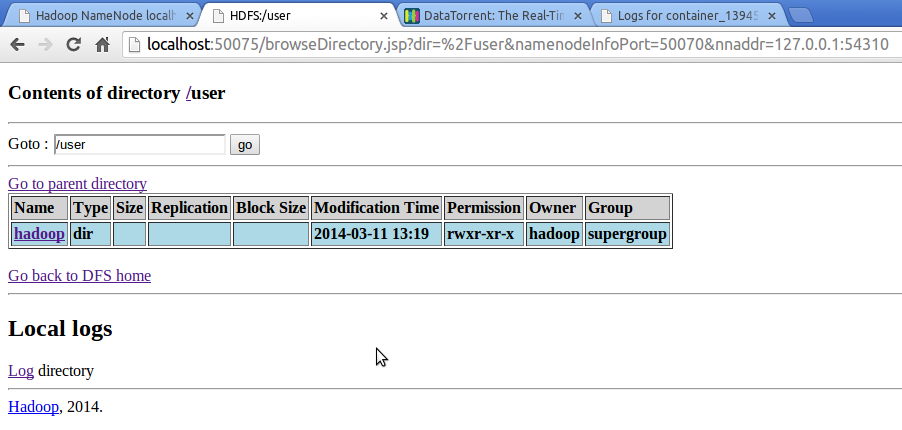


Figure HDFS Directory /usr

Click on the link “hadoop” should bring up the following UI that displays the HDFS files and directories that we are looking for,

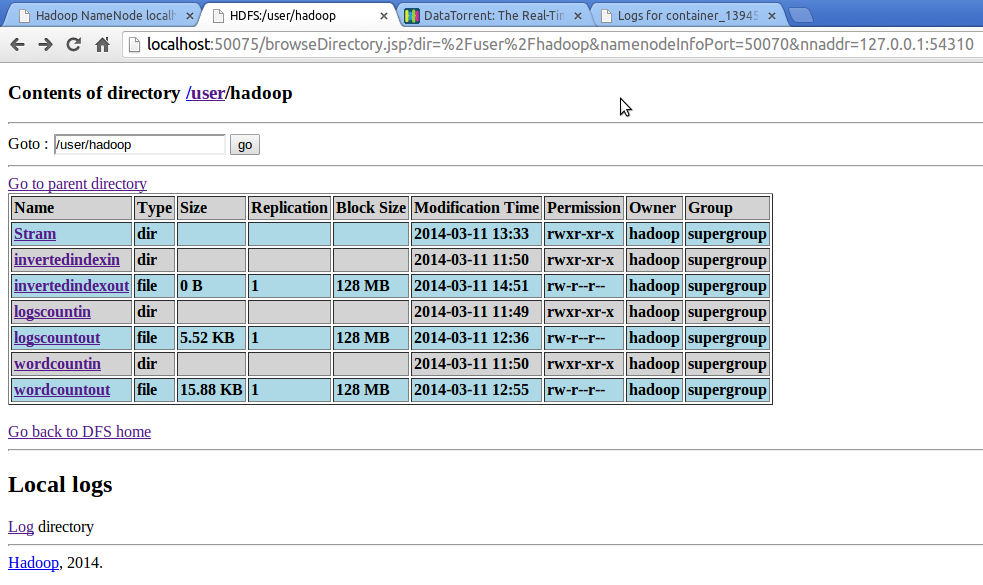


Figure HDFS /usr/Hadoop

Above screenshot displays various files and directories under HDFS. Assuming that the results for Word Count Application are written to HDFS output file – “wordcountout”, the contents of the file can be viewed by clicking on the link. This should display the file contents as shown in below screenshot,



Figure Word Count Results - "/usr/hadoop/wordcountout"

### Using HDFS commands

The Word Count results can also be viewed by using HDFS commands by copying the generated file from under HDFS to local file system. Following command can be used to do that,

$ hadoop fs –copyToLocal wordcountout .

The above command will copy the “wordcountout” file from HDFS location “/user/hadoop” to the current directory. Please take a look at following screenshot,

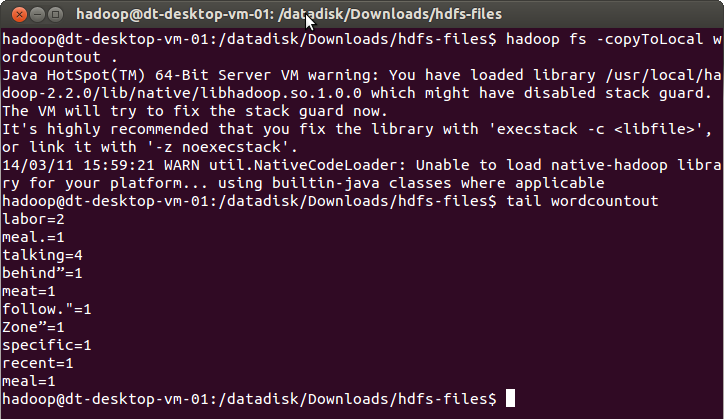


Figure Copying wordcountout file from HDFS to local file system

# Glossary of Terms used

###### Apache Hadoop

Apache Hadoop is the de-facto distributed operating system, details of which can be found on the web at - <http://apache.hadoop.org/>

###### HDFS

The Hadoop Distributed File System (HDFS) is a distributed file system. It is highly fault-tolerant and is designed to be deployed on low-cost hardware. HDFS provides high throughput access to application data and is suitable for applications that have large data sets.

###### Demo Application

The real-time stream processing application packaged with platform binary. The demo application can be launched in the standalone mode or on the Hadoop cluster to understand more about DataTorrent platform and its features.

###### Streaming Window

A streaming window is a slice of time that encapsulates a set of tuples. The collection of these tuples constitutes a window data set, which is also called as an atomic micro-batch.

###### Sliding Application Window

Sliding window is computation that requires previous N streaming windows. After each streaming window the Nth past window is dropped and the new window is added to the computation.

###### Directed Acyclic Graph (DAG)

It is the logical representation of real-time stream processing application. The computational units within DAG are called as operators and the data-flow edges are called as data streams.

###### Operator

An entity that holds a computational logic to process the data tuples. It is part of a real-time stream processing application. The Operator computational logic gets executed inside a container.

###### Port

Each operator can have ports on which it can receive input data tuples and also output processed data tuples.

###### Stream

A stream consists of data tuples that flow from one port of an operator to another.

###### Logical Plan or DAG

Logical Plan is the DOM (data object model) created as operators and streams are added to the DAG. It is identical to “Directed Acyclic Graph”.

###### Physical Plan or DAG

It’s the physical representation of the Logical Plan of the application and is a blueprint of how the application will run on the DataTorrent Cluster’s physical container and nodes.

###### Data Tuples Processed

Number of data objects processed by a real-time stream processing applications

###### Data Tuples Emitted

Number of data objects emitted after the processing is complete by a real-time stream processing applications