Pivotal

PRODUCT DOCUMENTATION

Pivotal HD DataLoader

Version 2.0

Installation and User Guide

Rev: A05

Use of Open Source

This product may be distributed with open source code, licensed to you in accordance with the applicable open source license. If you would like a copy of any such source code, Pivotal will provide a copy of the source code that is required to be made available in accordance with the applicable open source license. Pivotal may charge reasonable shipping and handling charges for such distribution.

Revised November 2013

Pivotal DataLoader Installation and User Guide 2.0.3 - Contents

Preface

This guide provides information for Hadoop administrators and superusers responsible for administering a DataLoader system.

About Pivotal, Inc.

Greenplum is currently transitioning to a new corporate identity (Pivotal, Inc.). We estimate that this transition will be completed in 2013. During this transition, there will be some legacy instances of our former corporate identity (Greenplum) appearing in our products and documentation. If you have any questions or concerns, please do not hesitate to contact us through our web site: `http://gopivotal.com/about-pivotal/support`.

About This Guide

This guide provides information and instructions for configuring, maintaining and using Pivotal DataLoader. This guide is intended for system and database administrators responsible for managing DataLoader.

This guide assumes knowledge of Linux/UNIX system administration, database management systems, database administration, and XML.

This guide contains the following main sections:

- Chapter 1, "Before You Begin" explains the setup and concepts used by DataLoader.
- Chapter 2, "Installing and Configuring DataLoader" contains the installation instructions and configuration parameters.
- Chapter 3, "Using DataLoader" explains how to use the DataLoader functionality.
- Chapter 4, "Loading Files and Push Streams into HAWQ Using PXF" explains how to use DataLoader and PXF to load stream data into HAWQ.
- Appendix A, "Data Stores" describes the properties used to register Datastores.
- Appendix B, "Job Specification" gives the elements and attributes of the Jobspec.
- Appendix C, "DataLoader Copy Strategies" discusses the supported copy strategies.
- Appendix D, "Job Transfer Policy" covers the transfer policy options used in submitting jobs.
- Appendix E, "Supported Combinations of Job Copy Strategy and Transfer Policy" defines the applicable settings for submitting jobs.
- Appendix F, "DataLoader Command Line Reference" lists the commands available in the Command Line Interface (CLI).

- Appendix G, "Configuring Flume for push streaming" tells how to configure Flume for use with push streams.

The Appendices explain other concepts that may be useful in using DataLoader.

Document Conventions

The following conventions are used throughout the DataLoader documentation to help you identify certain types of information.

- Text Conventions
- Command Syntax Conventions

Text Conventions

Table 0.1 Text Conventions

Text Convention	Usage	Examples
bold	Button, menu, tab, page, and field names in GUI applications	Click **Cancel** to exit the page without saving your changes.
italics	New terms where they are defined Database objects, such as schema, table, or columns names	The *master instance* is the `postgres` process that accepts client connections. Catalog information for DataLoader resides in the *pg_catalog* schema.
`monospace`	File names and path names Programs and executables Command names and syntax Parameter names	Edit the `postgresql.conf` file. Use `gpstart` to start DataLoader.
`monospace italics`	Variable information within file paths and file names Variable information within command syntax	`/home/dladmin/`*`config_file`* `COPY` *`tablename`* `FROM '`*`filename`*`'`
`monospace bold`	Used to call attention to a particular part of a command, parameter, or code snippet.	Change the host name, port, and database name in the JDBC connection URL: `jdbc:postgresql://`**`host:5432/mydb`**
`UPPERCASE`	Environment variables SQL commands Keyboard keys	Make sure that the Java `/bin` directory is in your `$PATH`. `SELECT * FROM` *`my_table`*`;` Press `CTRL+C` to escape.

Command Syntax Conventions

Table 0.2 Command Syntax Conventions

Text Convention	Usage	Examples
{ }	Within command syntax, curly braces group related command options. Do not type the curly braces.	FROM { 'filename' | STDIN }
[ ]	Within command syntax, square brackets denote optional arguments. Do not type the brackets.	TRUNCATE [TABLE] name
. . .	Within command syntax, an ellipsis denotes repetition of a command, variable, or option. Do not type the ellipsis.	DROP TABLE name [, ...]
|	Within command syntax, the pipe symbol denotes an "OR" relationship. Do not type the pipe symbol.	VACUUM [FULL | FREEZE]
$ system_command # root_system_command => gpdb_command =# su_gpdb_command	Denotes a command prompt - do not type the prompt symbol. $ and # denote terminal command prompts. => and =# denote DataLoader interactive program command prompts (psql or gpssh, for example).	$ createdb mydatabase # chown dladmin -R /datadir => SELECT * FROM mytable; =# SELECT * FROM pg_database;

Getting Support

EMC support, product, and licensing information can be obtained as follows.

Product information

For technical support, documentation, release notes, software updates, or for information about Pivotal products, licensing, and services, go to http://www.gopivotal.com.

Additionally you can still obtain product and support information from the EMC Support Site at: http://support.EMC.com.

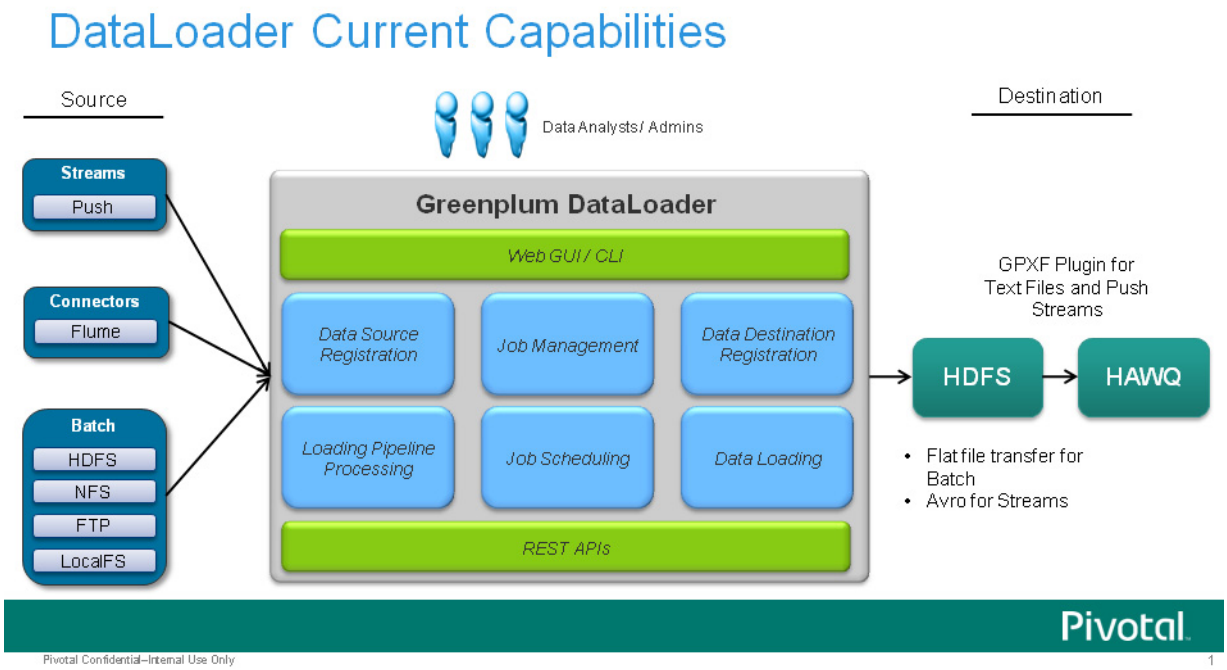
1. Before You Begin

The following notes and concepts are important to understand before beginning DataLoader Installation.

Overview of Pivotal DataLoader

Pivotal DataLoader is an advanced Big Data ingesting tool. It focuses on loading Big Data into Hadoop clusters. It is an enterprise solution for staged, batch data-loading for offline data analytics as well as realtime data streaming for online incremental data analytics. It also allows easy migration of data between large data cluster deployments.

DataLoader leverages Hadoop MapReduce to run loading jobs as a set of Map tasks. DataLoader can dynamically scale the execution of data loading tasks to maximize the system resource. With single node deployment, it linearly scales out on disk numbers up to the maximum machine bandwidth. With multi-node cluster deployment, it linearly scales out on machine numbers up to the maximum network bandwidth. This horizontal scalability promises optimized, and best possible throughput.



Staged, batch data loading is useful when throughput, linear scalability, and resource efficiency are priorities. In batch mode, Pivotal DataLoader can efficiently load large volumes of data.

Real time data streaming is useful in cases where latency, reliability, availability and connectivity are desired. Pivotal DataLoader can load large numbers of data feeds in real time, with linear scalability support.

DataLoader loads data into HDFS. DataLoader provides PXF adaptors to import files and streams in text or Avro format into HAWQ for real-time querying. See Chapter 4, "Loading Files and Push Streams into HAWQ Using PXF" for more details.

DataLoader partitions data into chunks when needed, splits jobs into multiple tasks, schedules the tasks, and handles job failures. Source data locality and network topology is taken into account.

DataLoader provides an easy-to-use interface to:

- Configure, start, and manage loading jobs from the user interface
- Manage jobs and data streams from the user interface GUI or command line
- Monitor job progress
- Define transformations to be applied in the data loading pipeline

Pivotal DataLoader Components

DataLoader consists of the following components:

Table 1 DataLoader components

Component	Description
DataLoader Master	Provides an operational and administrative graphical user interface, job and task scheduling. It also provides REST programmatic interface for integration with other tools.
DataLoader CLI	A command line tool that interacts with DataLoader Manager to provide the command line access for loading job operation.
DataLoader Web UI	A basic web console which interacts with DataLoader Master to help user submit batch jobs.
DataLoader Worker	Tasks that can be spawned on multiple nodes for transfer of source data.

Supported Platforms

DataLoader 2.0:

- Red Hat Enterprise 6.1-64 bit and 6.2-64 bit
- CentOS 6.1-64 bit, 6.2-64 bit, 6.3-64 bit, and 6.4-64 bit
- Browser: Firefox version 21

DataLoader Capabilities

DataLoader provides the following features.

Distributed, Parallel Loading Engine

- DataLoader provides a “big data pipe” of “data lanes” for moving big data in bulk or as streams in parallel.
- DataLoader supports bulk/batch loading with high throughput for big data and streaming with low latency for fast data.

Job loading with linear scalability through Job Scheduler

- DataLoader leverages commodity hardware for linear scalability: the more machine resources, the higher the aggregated processing power and IO bandwidth.
- Resource scheduling provides best throughput and/or low latency
- Optional bandwidth throttling helps limit bandwidth congestion
- Copy strategies with source and destination optimization

Pluggable architecture to support multiple data stores

- DataLoader provides pluggable data stores and data source protocols.

Pluggable data transformation

- User-defined “data pipelines” with compression, encryption, filtering and related functions; pluggable through JAR files. Transformation is executed at the time of writing the data as part of the map task.

Web UI

The web interface allows you to:

- Configure, start, and manage loading jobs from the user interface
- Manage jobs and data streams from the user interface UI or command line
- Monitor job progress

User authentication

DataLoader allows multiple users to login and create data loading jobs. Access control is provided through Linux OS-level authentication.

Data import to HAWQ

DataLoader provides PXF adaptors to allow push stream data and text files to be loaded into HAWQ for real-time querying.

API integration points

DataLoader will provide RESTful APIs for integration with other ETL tools in the future. Currently, Flume integration is supported out of box.

Planning your loading architecture

This topic describes the Pivotal DataLoader components, and the RPMs included in the package and provides a brief overview of installation.

DataLoader consists of the following core components:

- DataLoader Service

This component includes manager service, scheduling and other key services.

- DataLoader CLI

  The DataLoader command line tool (CLI) allows users to access all DataLoader functionality.

DataLoader 2.0 features an extremely flexible deployment architecture. The CLI and service component can be installed together on one node or separately on different nodes. For example, you might install DataLoader service on server hardware, but run CLI service from a laptop that has a network connection to the Data Service node. DataLoader 2.0 can execute loading jobs locally on the service node, or run the loading job on a Hadoop cluster where HDFS, MapReduce and Zookeeper are available to use.

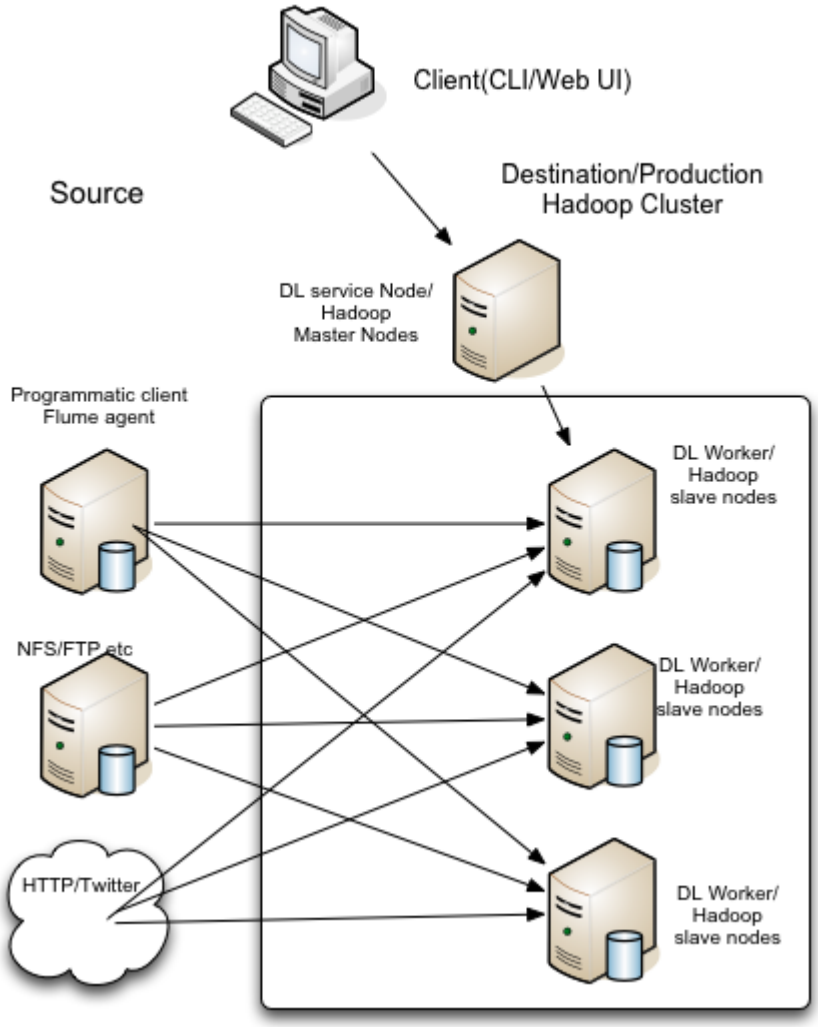
Depending on how DataLoader will be used, the following are typical scenarios supported by DataLoader:

Running DataLoader on a production Hadoop cluster

DataLoader Service can be installed on an edge node of the destination Hadoop cluster (1.x or 2.x). If the Hadoop DataNodes has connectivity with the data sources, DataLoader can leverage the Hadoop nodes to run Map "worker" tasks to load data in parallel. **No installation is needed for DataLoader worker nodes (map tasks); the required files are loaded into Hadoop DistributedCache for use at runtime.**

If there is limited external connectivity of Hadoop DataNodes (as in a DCA), DataLoader can run on the edge node in a pseudo-distributed mode to achieve parallel data transfer. In this case, the user must install a Hadoop cluster (1.x or 2.x) in pseudo-distributed mode on the edge node. For DCA, DataLoader will be installed in the DIA or mdw module.

The following shows a suggested network deployment for this configuration.



In this case, since the Pivotal HD cluster will be running other workloads as well, direct access to the local file system for data loading is not recommended.

Note: The DataLoader service node does not need to be co-located with the Hadoop Master Node.

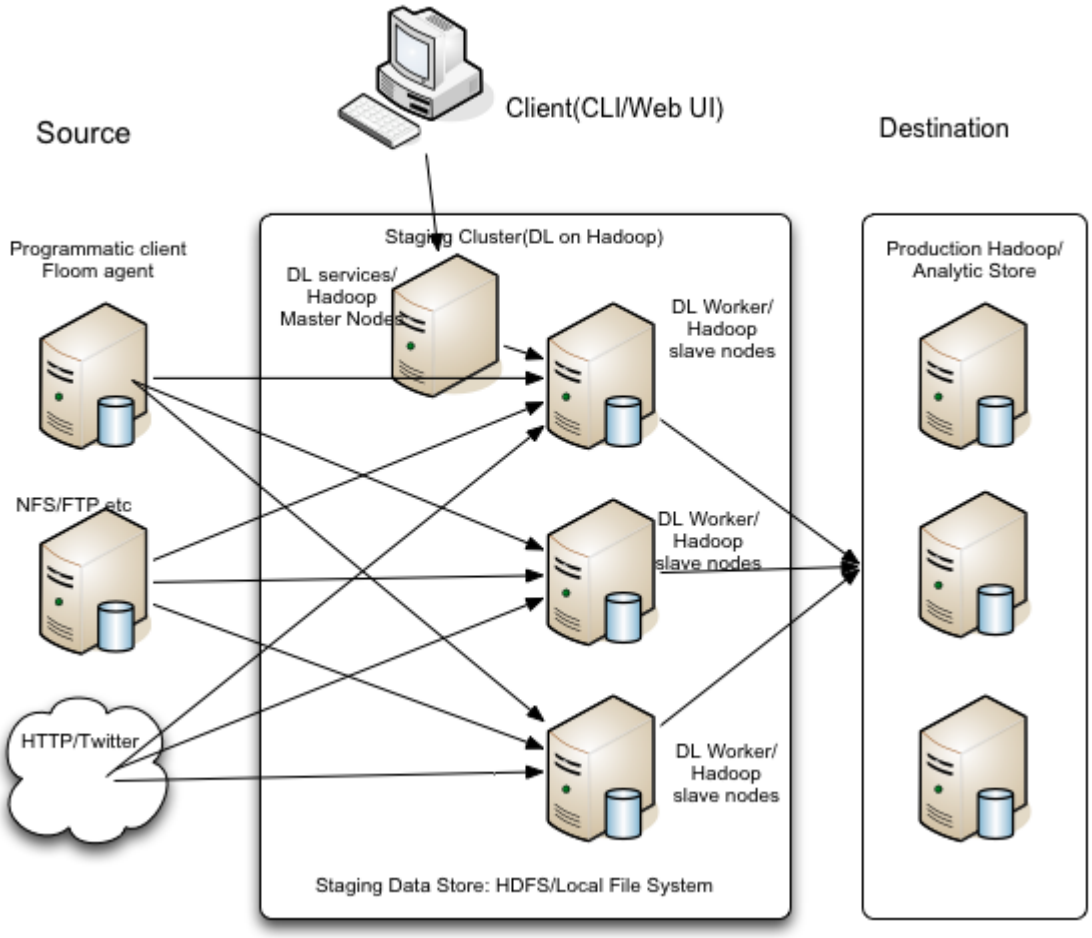
Suggested uses might be loading from a NFS/FTP server or a stream.

Running the DataLoader on the staged source cluster

When source data is on multiple disks, or protecting the production HDFS cluster performance is a priority, it is recommended to set up dedicated staging cluster where DataLoader can be co-located with other ETL jobs. DataLoader can be installed on the dedicated hardware to load data from the source such as FTP servers, NFS servers, HDFS, HTTP etc.

In this scenario, a Hadoop cluster is deployed on the staging nodes, with DataNodes on the nodes that have source data on local disks. DataLoader service is deployed on an edge node. No worker node installation is needed, as required files are loaded into Hadoop DistributedCache for use at runtime.

This scenario also applies to when the destination Hadoop cluster is in a DCA that has limited external connectivity. In such case, DataLoader can be deployed on the DIA or mdw module.



The staging cluster, using the storage disks that come with the hardware, provides data storage in following forms.

Local File System:

DataLoader can access the local file system directly. Users can mount disks to the servers in the cluster to load data directly from disks. DataLoader has additional optimizations to support loading from local disk, through LocalFS copy.

HDFS cluster:

DataLoader HDFS is installed on the staging cluster for use by DataLoader. HDFS can also be used as the interim storage to run MR/Pig/Hive jobs for ETL or other processing purposed before moving data into a production Hadoop cluster.

Use Staged Data Loading for:

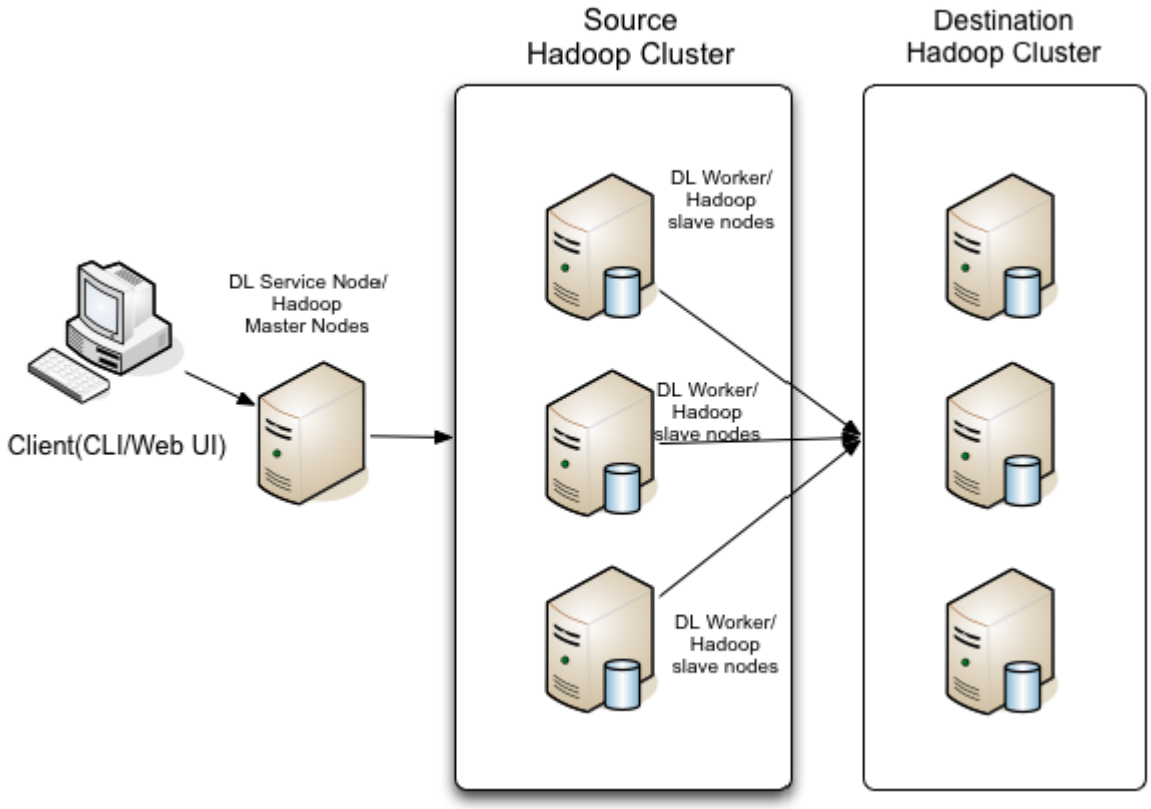
- Loading from NFS/FTP file stores

- Loading from staging cluster storage (HDFS, Local file system with data on mounted disk)
- Loading from data/event collection system (Flume, Http stream etc)
- Loading from backend/legacy systems via Data/Application integration frameworks/brokers such as databases, APIs and messaging systems.

Note: DataLoader provides an open Data Access Framework and API (data store framework/API) and supports data integration/API integration using Springframework and libraries (Springframework, Spring Integration, Spring Data etc.).

Cluster to Cluster Data Copy/Data Migration

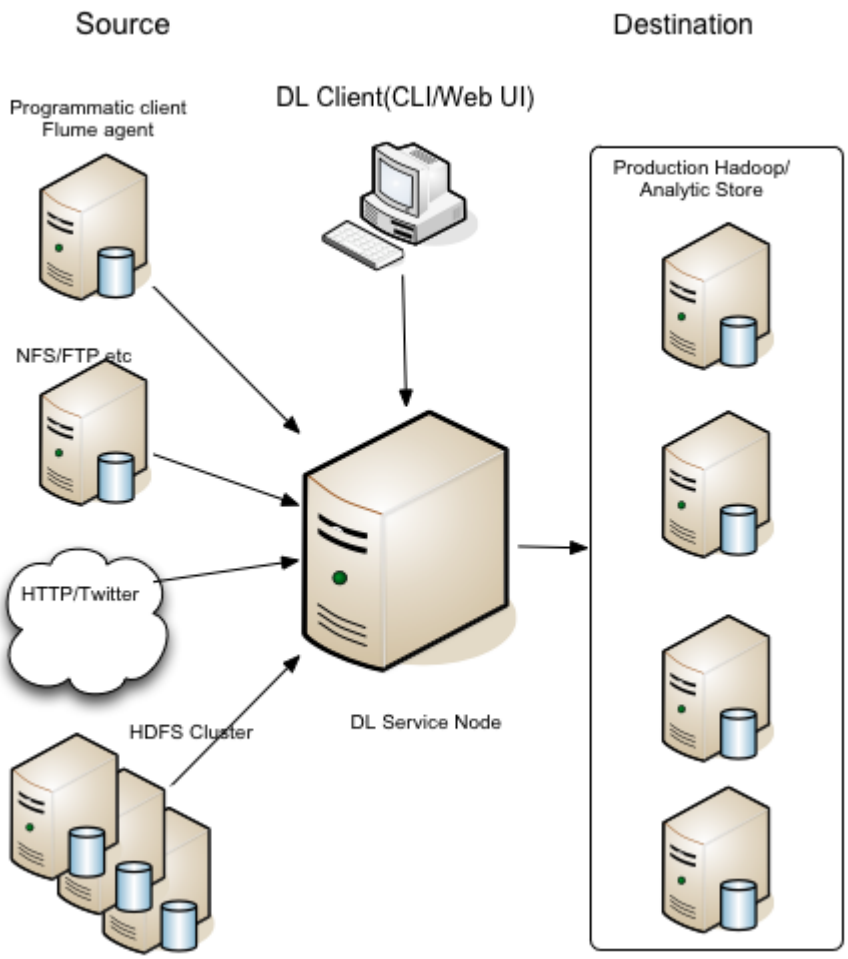
DataLoader can be used to copy HDFS data between Hadoop clusters. No dedicated hardware is needed: the DataLoader service node can be installed on any of the nodes. DataLoader jobs can be configured to run either on the source Hadoop cluster or destination Hadoop cluster. The following diagram shows the configuration for running DataLoader on the source Hadoop cluster.



Pseudo-distributed Mode

If you are not able to install DataLoader on a Pivotal HD cluster, you can use the pseudo-distributed mode on a single node. An example might be a DCA where DataNodes do not have external connectivity, so that DataLoader would need to be installed on a Master or DIA node. Pseudo-distributed mode will have a lower throughput than distributed mode.

A typical deployment for pseudo-distributed mode is shown below.



To increase loading speed, we recommend you deploy Hadoop on the single node with pseudo mode configuration. This will give DataLoader the capability to run multiple processes, and has each DataLoader loader process read from a single disk to maximize hardware usage. In this mode, first deploy Hadoop in pseudo-distributed mode, then deploy DataLoader on that node and configure as pseudo-distributed.

Data Availability

Failure Modes

DataLoader is built on a master-slave architecture. The master node consists of job management and scheduler services that use a database to store job data and Zookeeper for runtime state data. Scheduler uses YARN/MR1 for resource management and runtime job execution. DataLoader slaves are processes that are scheduled to run on a Hadoop cluster when a job is submitted.

Failures can be one of two types.

- Hadoop cluster failures:

  DataLoader needs Hadoop Mapreduce/HDFS and Zookeeper to be available to run DataLoader jobs. Any failure of these services can cause DataLoader job failure. In Hadoop 1.0 and 2.0, Jobtracker/Yarn and NameNode have single points failure. Please refer to the related documentation on high availability features and configurations of these components and services.

- DataLoader component failures:

  DataLoader master runs on a single node but also keeps a very light weight job state on the local filesystem. The runtime state is kept in Zookeeper, which is a highly available system. After a job is submitted, the existing batch job and push streaming jobs continue to run without master node intervention. (This feature is not yet available for push and pull streaming.)

Failure Recovery

When Hadoop failure occurs, DataLoader detects the failure and places the job into failed state. Jobs can be resumed after the Hadoop cluster is back online. For Hadoop failure recovery, refer to the Hadoop documentation.

Use of highly available system hardware can protect against job data loss caused by disk failure. If the data is still available after system restart, you can then simply restart DataLoader master service to automatically recover the jobs from last run.

Batch Loading Data Availabilty

In batch file copy, DataLoader relies on Hadoop high availability and does not store interim user data. Thus, when errors occur, there are no data loss or availability issues. When system and master services are restarted, the DataLoader master recovers the job state from the database and Zookeeper. If the master should fail, jobs continue to run, as they are map tasks.

Streaming Job Data Availability

Availability is achieved through 1) reliable event transfer between the DataLoader client and worker, and 2) making the stream data durable via checkpointing.

Reliable Event Transfer

The push stream client first contacts the DataLoader cluster, selects the least loaded worker, and establishes a push stream connection between the client and the selected worker. The client then sends the event data to the worker.

After the event data have been received by the worker and checkpointed on disk, the worker delivers the acknowledgment (ACK) back to the client in asynchronously mode thus the client is not blocked by the worker. All ACKs are queued on the worker side in a distributed, persistence queue backed by the Zookeeper cluster so ACKs are not lost even if there is a network or client failure. The ACKs are piggybacked to the client when the client contacts the DataLoader cluster. The client is responsible for ensuring all the ACKs are received for the events it tends to send. The client consider the event to be reliably sent to DataLoader only when the ACK has been received by the client.

For a given event, the event RPC API is idempotent, i.e. the client can repeatedly send the same event many times, but the worker only accepts the event once. The duplicated event received by DataLoader workers are dropped silently. The client can send an empty event to query all pending ACKs to check whether an event has been durably received by DataLoader workers.

Event Data Durability

The worker receives the data and store it in an internal memory buffer. The worker starts to checkpoint the event data in memory to disk when any of the following conditions are met:

1. The event data size stored in memory exceeds the configured threshold.
2. The number of events stored in memory has exceeded the configured number of messages.
3. The elapsed time since last checkpointing has exceeded the configured checkpointing interval.

The DataLoader worker stores checkpointing data on HDFS so the data can be accessed by all workers reliably. For all the events for which the client has received ACKs, the event data is considered durable and is guaranteed to be sent to the destination even in the face of worker failure.

Worker Failure Handling

During the push stream job, if any worker fails, the streaming client library detects the failure and it automatically redirects the client to make a connection to another active worker and continue to send data to the new worker. Concurently, the push stream job scheduler launches another worker instance to take over the event data received by the failed worker. The data left by the failed worker includes the following:

1. The data checkpointed on the shared storage.
2. Undelivered ACK in the persistent queue.

The new worker first processes the data by sending the events to the destination and managing the ACK queue. After the left-over event messages are processed, it opens new client connections.

In the event of a master failure, the user restarts the master services; the workers continue to transfer data independent of the master as the workers are managed by the Hadoop cluster, to be specifically MapReduce as it stands today.

2. Installing and Configuring DataLoader

DataLoader can manage a dynamic pool of loader machines for fast ingestion of big data. DataLoader works with both Hadoop 1.x (MR1) and Hadoop 2.x (Yarn) and can leverage them as the resource manager and job scheduler.

Note: A properly-configured Hadoop cluster is required, and HDFS, MapReduce, and Zookeeper must be available for DataLoader to use. For Hadoop/Zookeeper deployment instructions using the Pivotal HD distribution, refer to the *Pivotal HD Installation and Administrator Guide*.

This chapter covers installation and configuration of the RPM packages. For binary tarball installation, refer to Appendix H, "Installing and Configuring DataLoader from binaries".

Packages and Installers

RPM packages:

DataLoader uses two RPM packages that are deployed to the Client and Master, nodes, which can be the same. No worker installation is needed; required files are put in Hadoop DistributedCache for use at runtime.

Table 2.1 RPM Packages and Deployments

Package Name	Description	Deployment Nodes
dataloader-cli-2.0.4-nn.x86_64.rpm	dataloader-cli-2.0.4-nn.x86_64.rpm provides essential files to setup the DataLoader client.	Client node
dataloader-service-2.0.4-nn.x86_64.rpm	dataloader-service-2.0.4-nn.x86_64.rpm installs all needed components on the DataLoader master node	Master node

Note: The nn in Package Name is the RPM build number.

Installer Scripts

- install_dl.sh
- dl-cluster.conf.template

Prerequisites

Installation of User Account

In most customer environments, root account password and information is strictly controlled. To install DataLoader without using root account, use the following process.

Create an Installation User Account

1. Create a dladmin account on the machines where you will install DataLoader service and client. This dladmin account will be used as a system administrator account for installing DataLoader.
2. After the dladmin has been created, give it `sudo` permissions.

```
# vi /etc/sudoers
```

3. This will take you to a text editor with an `/etc/sudoers` file already opened. Add the following line:

```
dladmin ALL=(ALL) NOPASSWD: ALL
```

   and save the file.

Pivotal HD and Zookeeper Cluster Installation

To install Pivotal HD and the Zookeeper cluster, refer to the *Pivotal HD Enterprise Installation and Administrator Guide* for deployment instructions.

Make sure you include the following in your HDFS configuration

For HDFS 1.x and HDFS 2,x:

```
<configuration>
  <property>
    <name>dfs.support.append</name>
    <value>true</value>
  </property>
</configuration>
```

Install the JDK

1. Install the JDK: Download and install the Oracle JDK1.6 (Java SE6 or JDK 6) from:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

(http://www.oracle.com/technetwork/java/archive-139210.html)

2. After installing JDK, set the JAVA_HOME environment variable referring to where you installed JDK. On a typical Linux installation with Oracle JDK 1.6, the value of this variable should be `/usr/java/default/`.

```
export JAVA_HOME=/usr/java/default/
```

3. Add $JAVA_HOME/bin into your PATH environment variable. On a Linux platform with bash shell, add the following lines into the file ~/.bashrc:

```
export PATH=$JAVA_HOME/bin:$PATH
```

Download and extract the DataLoader files

Download and copy the PHDTools stack to /home/dladmin/ on the host where you want to install DataLoader. Make sure the Tarball has read permission for the user 'dladmin'.

Extract the tarball:

```
[root@hdp2-w17 dladmin]# chown dladmin:dladmin PHDTools-1.0.1-xx.tar.gz
[root@hdp2-w17 dladmin]# ls -lrt PHDTools-1.0.1-xx.tar.gz
-rw-r--r-- 1 dladmin dladmin 246080597 Jun 24 11:23 PHDTools-1.0.3-xx.tar.gz
[root@hdp2-w17 dladmin]# sudo su - dladmin
[dladmin@hdp2-w17 ~]$ tar xzvf PHDTools-1.0.1-xx.tar.gz
[dladmin@hdp2-w17 ~]$ ls -lrt PHDTools-1.0.1-xx
total 12
drwxrwxr-x 3 dladmin dladmin 4096 Jun 24 11:32 uss
drwxrwxr-x 3 dladmin dladmin 4096 Jun 24 11:32 dataloader
drwxrwxr-x 3 dladmin dladmin 4096 Jun 24 11:33 spring-data-hadoop
[dladmin@hdp2-w17 ~]$ cd PHDTools-1.0.1-xx/dataloader/rpm/
[dladmin@hdp2-w17 rpm]$ ls -lrt
total 213224
-rw-r--r-- 1 dladmin dladmin 199662368 Jun 24 11:32
dataloader-service-2.0.4-35.x86_64.rpm
-rw-r--r-- 1 dladmin dladmin  18647092 Jun 24 11:32 dataloader-cli-2.0.4-35.x86_64.rpm
-r-xr-xr-x 1 dladmin dladmin      5251 Jun 24 11:33 install_dl.sh
-r--r--r-- 1 dladmin dladmin       167 Jun 24 11:33 dl-cluster.conf.template
-rw-rw-r-- 1 dladmin dladmin        69 Jun 24 11:33
dataloader-cli-2.0.4-35.x86_64.rpm.md5
-rw-rw-r-- 1 dladmin dladmin        48 Jun 24 11:33 install_dl.sh.md5
-rw-rw-r-- 1 dladmin dladmin        59 Jun 24 11:33 dl-cluster.conf.template.md5
-rw-rw-r-- 1 dladmin dladmin        73 Jun 24 11:33
dataloader-service-2.0.4-35.x86_64.rpm.md5
```

Note: xx in the Package Name is the package build number.

Install DataLoader

Run following commands as user "dladmin" or any user that has root privileges:.

```
sudo [dladmin@hdp2-w17 ~]./install_dl.sh -i
```

DataLoader service and the CLI tool will be installed at the default location /usr/local/gphd/dataloader-2.0.4/.

After installation, DataLoader can be started in pseudo distributed mode without any further configuration.

Uninstalling DataLoader

There are two uninstall options. One preserves customer data, and the other uninstalls both DataLoader and customer data.

To uninstall DataLoader but preserve customer data, use the command:

```
[dladmin@hdp2-w17 ~]./install_dl.sh -u
```

To uninstall DataLoader and remove both binary and customer data including configuration, enter the command

```
[dladmin@hdp2-w17 ~]./install_dl.sh -u -c
```

Configuring DataLoader

Modify the file `dataloader-env.sh` to reflect your Pivotal HD and Zookeeper cluster configuration. Either host names or their IP addresses can be used. This should be done as root (enter command proceeded by `sudo`.)

The file is located in `/usr/local/gphd/dataloader-2.0.4/conf`.

The following table gives the parameters that should be modified.

Table 2.2 DataLoader Basic Parameters

Parameter	Function	Default value	Options
DATALOADER_START_MODE	This parameter controls the start mode of the DataLoader. Change this parameter if you want to start DataLoader in distributed mode; Note: If you wish to start DataLoader in pseudo distributed mode, please do not change the parameters.	pseudo	pseudo, distributed
DATALOADER_ZK_QUORUM	List of Zookeeper nodes in comma separated format. This list must be the same as used in Zookeeper cluster configuration, (in the “zk.cfg” file).	Must be specified.	N/A
DATALOADER_SERVICE_IP	IP address the services will bind to ("localhost" cannot be used).	Must be specified except for when running in standalone mode, and CLI is installed with service node	N/A
EXECUTOR_TYPE	Hadoop cluster version that DataLoader is running on. DataLoader supports Hadoop 1.0 (GPHD 1.2) and Pivotal HD 1.0.	mrv2	mrv1/mrv2
HADOOP_INSTALL	The Hadoop installation directory. It is used in DataLoader to set right classpath, user must make sure this value is set correctly	${HADOOP_INSTALL} If user does not export this shell env variable or value is empty, it must be specified manually	N/A
HADOOP_CONF_DIR	The Hadoop configuration directory.If you are using install_worker in tar ball to install worker, this value must be set. Otherwise optional	${HADOOP_CONF_DIR }If user does not export this shell env variable or value is empty, it must be specified manually	
DATALOADER_METRICS_REPORTING	Whether metrics report should be enabled. If not enabled, progress information will not be available.	true	true, false
DATALOADER_EXECUTOR_CAPACITY	The total number of slots(MRv1) or containers (MRv2) that are available to DataLoader. This number, called “workers,” is set according to expected number of stream jobs, batch jobs, and capacity in the HDFS cluster. As an example, the number would be 20 on a 10 node Hadoop cluster to support up to 10 stream jobs, and leave capacity for copying 10s of TB of data in multiple simultaneous batch jobs. You must set this value for DataLoader to run correctly.	No default value, must be specified except for standalone mode.	

Table 2.2 DataLoader Basic Parameters

Parameter	Function	Default value	Options
DATALOADER_SYSTEM_DIR	This is a value pointer to a directory on the executor's corresponding HDFS. This folder is used by DataLoader as its working directory; it should be writable by the "dataloader" user; This folder should be created and grant edwrite access to the "dataloader" user attempting to start DataLoader service in distributed mode.	/dataloader/	N/A
DATALOADER_DATA_DIR	The location to put DataLoader data files. This parameter is set during the pre-installation process using dl_cluster. conf.	For RPM installation, it is set in /etc/default/dataloader, value is: /var/lib/gphd/dataloader. In other case, it is ${DATALOADER_HOME }/data	
DATALOADER_LOG_DIR	The location to put the DataLoader log files. This parameter is set during the pre-installation process using dl_cluster. conf.	For RPM installation, it is set in /etc/default/dataloader, value is: /var/log/gphd/dataloader. In other case, it is set to: ${DATALOADER_HOME }/log	

Do not modify the following parameters.

- DATALOADER_HOME
- DATALOADER_CONF_DIR
- ZK_RUNTIME_DIR
- ENGINE_HOME

Start and Stop DataLoader services

Before starting DataLoader services, change file ownership for the Linux account login. Change the permission of the file /etc/shadow to be readable by the group root

```
root# chmod g+r /etc/shadow
```

Start DataLoader manager directly using startup script

Login to the DataLoader master node as dladmin, and go to the /bin directory of the DataLoader installed location:

```
$su - dladmin
$cd /usr/local/gphd/dataloader-2.0.1/bin/
```

Note: To reset the default `dlamin` user password, login as root and use "passwd dladmin" to change the password.

To start DataLoader service:

```
$sudo -u dataloader ./dataloader.sh start
```

Note: To control DataLoader to start in pseudo/distributed mode, please change the "DATALOADER_START_MODE" property in the dataloader-env.sh file.

To stop DataLoader service:

```
$sudo -u dataloader ./dataloader.sh stop
```

Note: In the above command, dataloader is a service the user created during installation.

Using the Linux service utility to start/stop DataLoader services

You can also use the Linux service utility to start and stop DataLoader services.

1. To start DataLoader services

```
$ sudo service dataloader-manager start
```

2. To stop DataLoader services

```
$ sudo service dataloader-manager stop
```

Advanced Configurations

These configurations are used for performance tuning and are found in the `/usr/local/gphd/dataloader-2.0.4/conf/dataloader.xml` file.

Table 2.3 Advanced Configuration Values

Parameter	Explanation	Default value
dataloader.zk.address	Comma separated host/port list of zookeeper, this value should be changed, please change the DATALOADER_ZK_QUORUM in the *dataloader-env.sh* file.	${DATALOADER_ZK_QUORUM}
dataloader.manager.service.port	The manager for the restful interface listening port;	12380

Table 2.3 Advanced Configuration Values

Parameter	Explanation	Default value
dataloader.metrics.monitoring.enable	This flag specifies whether to enable the metrics reporting mechanism for DataLoader. Since progress monitoring depends on metrics reporting, disabling this parameter will also disable progress reporting. Changing the value of DATALOADER_METRICS_REPORTING in dataloader-env.sh will change the setting for this entry.	${DATALOADER_METRICS_REPORTING}
dataloader.metrics.server.host	rpc host address for metrics reporting server. This parameter should be set to the manager's host. This value should not be changed, if you want to change this value, please modify DATALOADER_SERVICE_IP in the *dataloader-env.sh* file.	${DATALOADER_SERVICE_IP}
dataloader.metrics.server.port	rpc port for metrics reporting server	12322
dataloader.scheduler.service.rest.port	The scheduler's restful interface listening port	12321
dataloader.scheduler.service.rest.host *	The scheduler restful interface binding address. This value should not be changed, if you want to change this value, please change DATALOADER_SERVICE_IP in the *dataloader-env.sh* file.	${DATALOADER_SERVICE_IP}
dataloader.scheduler.taskscheduler.host	The scheduler's host. The worker will use this to contact scheduler for runtime task scheduling. This value should not be changed, if you want to change this value, please change DATALOADER_SERVICE_IP in the *dataloader-env.sh* file.	${DATALOADER_SERVICE_IP}
dataloader.scheduler.taskscheduler.port	The scheduler's port. The worker will use this to contact the scheduler for runtime task scheduling,	12320
dataloader.worker.reader.num	The number of reader threads to be started in each worker. Reader threads are responsible for reading data from data sources. Using multiple reader threads can enhance I/O read parallelism. However, adding excessive reader threads can burden system resources due to the context switch between reader threads.	3

Table 2.3 Advanced Configuration Values

Parameter	Explanation	Default value
dataloader.worker.writer-pipeline.num	The number of writer/pipeline threads that will be started in each worker. Writer threads are responsible for writing data to its destination. Using multiple writer threads can enhance I/O write parallelism. However, adding excessive writer threads can burden system resources due to the context switch between writer threads.	5
dataloader.worker.buffer.num	The number of buffers which could be used in each worker for batch jobs. When all the buffers run out of DataLoader worker resources, the reader threads wait for buffers to become available, before reading next piece of data. Increasing the number of memory buffers used by DataLoader workers can potentially enhance the throughput and latency of a worker assigned to a job's workload. However, this strategy will use more system resources.	12
dataloader.worker.buffer.size	The size of each buffer for batch job workers. A reader thread first fetches a buffer of the specified size, before reading data from the source. It fills the buffer until either the buffer is full or the designated piece of data is completely read into buffer, then passwa it to the writer thread, to write it to the destination. Increasing the size of the memory buffers used by DataLoader workers can potentially enhance IO throughput, since it allows data read to occur in batch mode, reducing the number of I/O operations. However, it will use more system resources.	16 * 1024 * 1024 (16MB)
dataloader.worker.streaming.server.memory.upper.limit	The maxiumum memory that the worker can use to hold incloming data in pushstream jobs. When the size of event data held in a worker's memory not written to its destination exceeds this memory limit, the worker will stop receiving event data from the client, to avoid memory leakage. Increasing the value of this parameter can potentially enhance the latency of pushstream jobs when a worker is receiving large amounts of data from many clients. However, it will use more system resources.	335544320

Table 2.3 Advanced Configuration Values

Parameter	Explanation	Default value
dataloader.executor.notification.host	The IP address on which the executor is to listen for the job status notification: ${DATALOADER_SERVICE_IP}	
dataloader.executor.notification.port	The port on which the executor is to listen for job status notifications: 12324	12324

Advanced features are used primarily for performance tuning.

Configure DataLoader CLI

To use the DataLoader CLI for managing jobs and datastores, it must be configured. The configuration file for CLI is located at:

`${dataloader_HOME}/conf/dataloader-cli.conf.`

It is a text file with key/value pair content and contains the following values, shown below.

Table 2.4 CLI Configuration file values

Name	default values	Description
dataloader.api.url	http://localhost:12380/manager	This is the location of the DataLoader manager web address. If CLI is installed on the same machine as the DataLoader service package, this value doesn't need to be changed
dataloader.zk.address	localhost:12181	This is the location of the Zookeeper servers, provided as a comma-separated list. This list should be the same as used in dataloader-env.sh earlier. Do not change if using standalone or pseudo-distributed modes, as the embedded-zookeeper address is the default.
dataloader.cli.queue.max.buffer	32768	The maximum number of unacknowledged messages that the client can hold in memory. Will be overwritten at run time if the command line param --queue-size is specified. To avoid memory leakage, when the number of unacknowledged events/messages held in the client's memory exceeds the specified limits, the client will stop accepting events from the stream until the resource become available. Increasing the value of this parameter can potentially enhance the latency when a client is receiving numerous events. However, this will use more system resources.

If using pseudo-distributed mode, you can use default values, provided you are not using an external Zookeeper.

3. Using DataLoader

The following sections describe how to use DataLoader.

Overall Workflow

You can use the command line tool or Web GUI to perform the tasks listed below (in order of user activities). The Web GUI currently only supports batch jobs; streaming jobs must be started through the CLI. Once started, streaming jobs can be managed through the Web GUI.

Note: You must be logged in to have execution access.

You can:

- Register sources and destination as data stores
- Create a data-loading job
  - Specify a data store as Source
  - Specify a data store (HDFS for GPHD 1.2, HDFS2 for Pivotal HD) as Destination
  - Select Chunking / Compressing data during transfer
  - Select the Copy Strategy
  - Insert any other processing JARs, e.g., encryption
- Submit a Job
- Query a job
- List jobs
- Monitor job progress

By default, an anonymous user has only read-only access to the Web GUI.

A user must be logged in to perform job/datastore operations. A user can only suspend/resume/cancel jobs that are submitted by this user.

For a user to perform these functions, use the following steps:

1. Create a Linux user account on the machine where DataLoader manager is installed.
2. Login as the user on the Web GUI or ssh as the user to the command line to get execution access.

Note: Users must have access to the source data.

3. Create a /home/username directory in the destination HDFS.

To be able to write data to HDFS, either a user home directory must be present in the destination HDFS, or the user must be part of the superuser group in the Hadoop cluster.

By default, dataloader creates a single "dladmin" user for this purpose. If using dladmin, a /home/dladmin directory needs to be created in destination hdfs or dladmin needs to be a superuser. If using dladmin, dladmin needs to have access to source data.

About job lifecycles

Batch job lifecycle

When a batch job is submitted, it is first in SUBMITTED state. Batch jobs automatically start loading data and change their job status to RUNNING. After all the data has successfully loaded to its destination, the job status changes to COMPLETED. If the job fails, the status changes to FAILED.

A batch job can be SUSPENDED. When suspended, the job retains the list of files that have already been copied. A suspended batch job can be RESUMED. When resumed, the job resumes execution and only copies the files that are left uncopied. When chunking is enabled, only those chunks still to be copied are copied after resuming a suspended job. Loading of chunks or files that were in the middle of transfer when the job was SUSPENDED is restarted.

A batch job can also be CANCELED. Once canceled, the job is put into CANCELED job state, and cannot be resumed.

Streaming job lifecycle

When a streaming job is submitted, it is first in SUBMITTED state. DataLoader scheduler first determines how many workers are needed to run this streaming job. For push streaming, since the number of clients that will connect to the workers is unknown, you must specify the number of workers (in the job transfer policy) when submitting the job. For pull streaming, scheduler will calculate the required number of workers to run the job, based on the number of feeds you specified. You can also specify the number of workers explicitly.

Once a job has successfully entered its SUBMITTED state, DataLoader service will start the job right away. The job changes to STARTED state. When the required number of workers are up and running, the job will change to the RUNNING state. In this state, for push stream jobs, the workers are ready to accept client connections; you must stop (cancel) the job explicitly to complete the job. For pull streaming jobs, after all the data has successfully streamed to its destination, the job status changes to COMPLETED. If the job fails, the status changes to FAILED.

A streaming job can be SUSPENDED. When suspended, all stream workers will be released. A suspended job can be RESUMED. When resumed, the job continues appending to the job storage files.

A job can also be CANCELED. Once canceled, the job is put into a canceled job state, and cannot be resumed.

Registering and unregistering data stores

Managing data stores through the DataLoader console

To load data from a data store, the data store must be registered with Pivotal DataLoader.

The easiest way to register a new data store is through the DataLoader console web interface.

Registering a Data Store

1. Go to the web page at http://<dataloader_service_hostname>:12380/manager

Note: The default port for the Web UI is 12380.This port may change according to your configuration.

2. Click on the upper right corner to log in.

3. Click on the **Data Stores** entry on the Task bar to enter the data store instance page.

   Click on **Register New Data Store** to bring up the New Data Store Instance page.

4. On the New Data Store Instance page, enter the type of datastore (HDFS2, LOCALFS, FTP, NFS, or HDFS), host, port, root path, username/password when required, and any additional properties.

   DataLoader maintains the source directory structure from the root directory when copying. Thus, you must set the root directory accordingly: for example, if the source directory is in /home/testuser, set the directory to that destination.

Notes:

   a. To register a NFS type data store, you should mount the NFS share to both the master machine and Hadoop tasktracker nodes (node-manager nodes for YARN). Also, all NFS nodes must mount to the same directory structure.

   b. LocalFS can only be used with local disk data on DataLoader worker nodes (Hadoop DataNodes). You can register any DataLoader slave machine in the Hadoop cluster as a localFS type data store.

   c. When using DataLoader in distributed mode, for LocalFS data stores, you must change the Fairscheduler with DataLoader's modified scheduler. The cluster must be restarted for HDFS 2.x. See Appendix A, “Data Stores” for details.

   d. To register an HDFS datastore, the host is namenode, and the port is typically 8020.

5. If you need to specify more data store properties select **Add Property**. See Appendix A, “Data Stores” for more information.

6. Click the **Create** button on the page to complete the registration. The new data store will appear on the Data Store Instances page. if you entered invalid information, you will be warned that the datastore cannot be registered.

Supported Datastore Types

DataLoader supports the following datastores.

- nfs
- ftp
- localFS
- GPHD1.2
- Pivotal HD 1.0
- Apache Hadoop 1.0.3
- Apache Hadoop 2.0.2-alpha

Regarding datastore registration properties, please see Appendix A, "Data Stores" for more information.

Un-registering or Deleting a Data Store

Log in, if not already logged in.

1. Click on the **Data Stores** tab to see the Data Store Instances page.
2. Click the **Remove** button of the data store you want to remove.

Managing a data store through the command line

To register a data store using the command line

Note: Perform the following command line operations on the DataLoader CLI machine.

1. On the Client node, create the property file for the datastore as <propertyFile>.

   DataLoader maintains the source directory structure from the root directory when copying. Thus, you must set the root directory accordingly: i.e., if the source directory is in /home/testuser, set the directory to that destination.

   The following is an example of a property file.

```
type=localfs
host=<hostname>
rootpath=/
username=test_user
password=test_user
```

Note: Datastore sample property files are not included in the RPM package. For information regarding datastore registration properties, refer to Appendix A, "Data Stores" for more information.

2. Run the following command to register the data store in CLI node.

```
$ cd /usr/local/gphd/dataloader-2.0.4/bin/
$ ./dataloader-cli.sh config --command register -i <propertyfile>
```

3. Create a <propertyfile> for the datastore as shown in Appendix A, "Data Stores". For example, to register an nfs datastore, the commands would be:

```
$ cd /usr/local/gphd/dataloader-2.0.4/bin
$ ./dataloader-cli.sh config --command register -i nfs.property
```

Note: The host must be the hostname of the node.

Un-registering or Deleting a Data Store through the Command Line

1. Run the following command to list all the registered datastore IDs.

```
$ cd /usr/local/gphd/dataloader-2.0.4/bin
$ ./dataloader-cli.sh config --command list
```

2. Run the following command to delete the datastore.

```
$ ./dataloader-cli.sh config --command delete -n datastore_ID
```

Managing Jobs through DataLoader Web Console

Batch jobs can be managed through DataLoader Web Console.

Open a browser window with the URL:

`http://<dataloader_service_hostname>:12380/manager/`

The default port for the Web UI is 12380.

Creating and submitting a Job

Note: Currently, only batch jobs can be created and submitted this way. Stream jobs must be submitted through the CLI.

1. Click on the upper right corner to log in.
2. Select **Create a New Job** on the Home page.

You can submit your job using either basic or advanced properties.

3. To submit using the basic option, provide the required values shown in Basic property values for submitting a job. You will need to specify the type of source datastore. The available datastores will appear in the Source Path. Click on the green button to add a source datastore. It will appear in the **`Selected`** field.
4. The available targets are shown in the Target Path field. Click to select a target.
5. The default copy strategy is Intelligent, which selects the best strategy, based on the data being loaded. For more information, see Appendix C, "DataLoader Copy Strategies".

Basic Property Values for Submitting a Job

Property	Description
Source Datastore	the source data store hostname and port.
Source Path	the path of the data you want to load in source data store.
Target URI	the destination data store hostname and port.
Target Path	the path of the data you want to copy to in destination data store

To submit using advanced options, click **Show Advanced Options**. Refer to Appendix D, "Job Transfer Policy" for more information on the Advanced options.

Advanced Property Values for Submitting a Job

Property	Description
Worker Number	Number of workers to start for this job
Bandwidth Throttling	Enable throttling at the Bandwidth setting.
Bandwidth	Maximum bandwidth used before throttling is enabled.
Chunking	Use chunking when conditions in Chunking Threshold and Chunking size are met. Cannot be used if Compression is enabled.
Chunking Threshold	96M default
Chunking size	64M default
Overwrite existing file	Default is not to overwrite
Compression	Compress payload for transfer using Snappy. Cannot be used if Chunking is enabled.
Workers per disk	The number of workers that will copy the files in a single disk.

6. When you are done, click **`Submit`**. Pivotal Data Loader uses the default value for optional fields. After submitting the job, you can search under the Running Jobs list in the home page
7. You can get detailed information about the job by clicking on the job ID.

Monitoring a Job

Click the job ID to monitor details and find detailed job information.

You can check the progress bar (for batch jobs) or see the active workers (for streaming jobs).

Note: If files do not refresh, clear the browser cache

Suspending a Job

1. Click on the upper right corner to log in.
2. Click **Home** to go to the Home page.
3. In the Running Job list on the Home page, find the job ID.
4. Click the Job ID to show the Job Detail page.
5. Click **Suspend**. DataLoader will warn you that the Suspend command was sent, and will change to SUSPENDED state.
6. The job will be moved to the Suspended job list. It can be restarted by clicking **Resume** on the Job Detail page.

Resuming a Job

1. Click on the upper right corner to log in.
2. In the Suspended Jobs list on the Home page, find the Job ID.
3. Click the Job ID to show the Job Detail page.
4. Select **Resume** in the Job Operations list. DataLoader will warn you that the Resume command has been sent and will change to RUNNING or STARTED state.
5. You can check the home page to confirm that the job is running again

Stopping a Job

1. Click on the upper right corner to log in.
2. On the home page, search the Running Jobs or the Suspended Jobs list to find the job you want to stop.
3. Select the Job ID to go to the Job Detail page.
4. From the Job Operation list, select the **Cancel** button. DataLoader will warn you that the Cancel command has been sent and will change to CANCELLED state.
5. Confirm that the job is listed in the Canceled Jobs list.

Managing jobs through the command line

Through command line, you can manage both batch job and streaming jobs.

Submitting a Job Through the Command Line

1. Prepare the Job Specification File

   You must prepare the transfer specification file to submit a job using the command line. The transfer specification file is an XML file that specifies the source and destination data store, and the file or folder to load. Refer to Appendix B, "Job Specification" for more information and job spec samples.

2. Submit the job through command line after the transfer specification file has been prepared. See Appendix F, "DataLoader Command Line Reference" for more information.

   Note: Some options in the Policy file can also be specified as a command argument. If this is the case, the Command Line argument will take precedence.

```
$ dataloader-cli.sh submit -i <job specification file> -s
<strategy> -p policy.property
```

An example policy file for batch jobs is shown below. You can find sample specs in the dataloader CLI installation directory: /usr/local/gphd/dataloader-{version}/sample/

```
WORKER_NUMBER=0
IS_BAND_WIDTH_THROTTLING=false
BAND_WIDTH=0
IS_CHUNKING=false
CHUNKING_THRESHOLD=100663296
CHUNKING_SIZE=67108864
IS_OVERWRITE=true
USE_COMPRESSION=false
STRATEGY=intelligent
STREAM_DATA_DURABILITY=false
WORKERS_PER_DISK=
```

List jobs with a specific state

List all jobs in some specific state:

```
$ dataloader-cli.sh list --status RUNNING
```

The above command lists all jobs in RUNNING state. You can also get jobs with other states.

Monitoring a Job Using Command Line

Check the job status and progress through the query command line option

```
$ dataloader-cli.sh query -n <jobid>
```

Suspending a Job Using Command Line

```
$ dataloader-cli.sh suspend -n <jobid>
```

To Resume a Job Using Command Line

```
$ dataloader-cli.sh resume -n <jobid>
```

To Stop a Job Using Command Line

```
$ dataloader-cli.sh cancel -n <jobid>
```

To Push data to DataLoader Streaming jobs

You must first create a push stream job as explained in last section. After the job is started, the DataLoader CLI can be used to push data to the push stream job. The sections below show the basic usage of push data. An end-to-end example is given in the section following. For more details about the CLI options, please refer to Appendix F, “DataLoader Command Line Reference”.

Example of a Push Stream Job Specification

Start a Push stream job using the submit command with following specification:

```
<JobSpec xmlns="http://www.pivotal.com/hd/dataloader">
<name>name</name>
<type>streaming</type>
<streaming type="push">
<destination>
<datastore>hdfs://YOUR_TARGET_NAMENODE:8020/</datastore>
<path>/tmp</path>
</destination>
<reader type="PUSH_STREAM_READER">
</reader>
<writer type="PUSH_STREAM_WRITER">
       <!-- rolling.size is 10KB -->
      <property name="rolling.size"    value="10240"/>
</writer>
</streaming>
</JobSpec>
```

This spec tells DataLoader to create a pushstream job; the received data will be stored to the HDFS datastore `hdfs://YOUR_TARGET_NAMENODE:8020/`, into the folder `/tmp`. The number of workers to start is specified in the transfer policy.

Rolling size should be set by default to 64MB, the HDFS block size, for maximum performance. For small streams or for demonstration purposes, set to a small number such as 10KB (10240).

To push data from single file that does not change

```
$ dataloader-cli.sh pcat -n <jobid> --source testdata.log --format TEXT
```

This command reads data from testdata.log, line-by-line, and push these to the specified dataloader streaming job. CLI will exit when the end of this file is reached.

To push data from single file that keeps changing

```
$ dataloader-cli.sh ptail -n <jobid> --source testdata.log --format TEXT --idle 10000
```

This command will read data from testdata.log, line-by-line, and push these to the specified dataloader streaming job. CLI will exit only if this testdata.log has not been changed for 10 seconds. If the --idle option is not set, CLI will keep polling the changes in this file until error is detected or the job is closed.

To push data from STDIN

```
$ dataloader-cli.sh pstream -n <jobid>
```

This command will read data from STDIN, line-by-line, and push these to the specified dataloader streaming job. CLI will exit when it receives an end-of-stream signal (ctrl+d).

One possible use case might be

```
$ cat testdata.log | dataloader-cli.sh pstream -n <jobid>
```

This pushes the contents of testdata.log to the specified job.

To push data from a specified folder

```
$ dataloader-cli.sh scan -n <jobid> --scanfolder /data/testdata/ --thread-pool-size 4
```

This command scans the folder /data/testdata for files and pushes them to the specified dataloader streaming job. CLI will start four threads to read the data concurrently. CLI will keeping scanning this folder for new files until it is explicitly stopped by a `Ctrl+c`. command.

The thread pool size setting determines the number of threads to startto read files from a specifiedfolder. Each thread will read data from a single file until all the data in the file is transferred to the destination, then the thread will move to next file in the folder.

Increasing the number of threads can potentially enhance parallel processing of I/O, provided that the destination worker has sufficient capacity.

Pushstream Example

In this example, the job is run from the command line.

Log into the DataLoader CLI node.

Create the file “file1” using below command

```
# dd if=/dev/zero of=/file1 bs=1024 count=1024
```

Register HDFS datastore:

1. Go to the web page at http://localhost:12380/manager and select the Data Stores button to enter the data store register page.
2. Select the Register New Data Store button to bring up the New Data Store Instance page.

3. Enter values in the New Data Store Instance page to register new data store.

```
type=hdfs
host=<hdfs namenode host>
port=<hdfs port>
rootpath=/
```

4. Click the Create button on the page to complete the registration. The new data store can be found in the Data Store Instances page.

Run the job via the command line:

1. Log into the DataLoader CLI node.
2. Prepare job specification "spec.xml" under /usr/local/gphd/dataloader-2.0.4/bin as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<JobSpec xmlns="http://www.pivotal.com/hd/dataloader">
<name>name</name>
<type>streaming</type>
<streaming type="push">
<destination>
<datastore>hdfs2://Your_TARGET_NAMENODE:8020/</datastore>
<path>/2013</path>
</destination>
<reader type="PUSH_STREAM_READER">
</reader>
<writer type="PUSH_STREAM_WRITER">
<!-- rolling.size is 10KB -->
<property name="rolling.size" value="10480"/>
</writer>
</streaming>
</JobSpec>
```

Prepare a policy file “policy.property” under /usr/local/gphd/dataloader-2.0.4/bin.

```
WORKER_NUMBER=<n>
IS_BAND_WIDTH_THROTTLING=false
BAND_WIDTH=0
USE_COMPRESSION=false
STRATEGY=pushstream
```

Specify the worker number to the number of DataLoader workers available in the cluster.

Execute following command to submit the job:

```
# cd /usr/local/gphd/dataloader-2.0.4/bin
# ./dataloader-cli.sh submit -i spec.xml -s pushstream –p
policy.property
```

The job is submitted and jobID is displayed on the console. Please note the JobID.

Push file to the job.

```
# cd /usr/local/gphd/dataloader-2.0.4/bin
# cat /file1 | sh dataloader-cli.sh pstream -n <JobID>
```

Once the file gets uploaded, the DataLoader job can be cancelled:

```
# cd /usr/local/gphd/dataloader-2.0.4/bin
# ./dataloader-cli.sh cancel –n <jodID>
```

Verify that the files were transferred to the destination FS.

Troubleshooting

Check the following DataLoader log files under directory: /var/log/gphd/dataloader/

- dataloader-manager.log
- dataloader-manager.out
- dataloader-scheduler.log
- dataloader-scheduler.out

The DataLoader log file for the command line interface is located in the directory where you run the commands: dataloader-cli.log

It is very helpful to check the Hadoop cluster log to track the status of the DataLoader workers. Check the Hadoop cluster log files under:
`/user/lib/gphd/hadoop/logs/`

Check the Hadoop NameNode monitoring page:

http://namenode.host:50070/dfshealth.jsp

The YARN cluster resource monitoring is located at:

http://resourcemanager.host:8088/cluster

4. Loading Files and Push Streams into HAWQ Using PXF

DataLoader writes push streams and text/Avro files into HDFS, using push stream clients. The data is written in text or Apache Avro format, which is exposed through DataLoader APIs. You can either run batch jobs like MapReduce to analyze the data, or utilize Pivotal HAWQ to perform fast SQL queries on the data.

This chapter describes how to configure HAWQ/PXF to import data that is written to HDFS by DataLoader.

Prerequisites

To perform this operation, you need a running HAWQ and PXF on top of Pivotal HD.

HAWQ is a powerful SQL engine running on top of Pivotal HD; PXF is a connector between HAWQ and Pivotal HD, which provides HAWQ the ability to read files on HDFS and perform SQL queries. For more details on setting up HAWQ and PXF, please refer to their related documentation.

Configure the DataLoader PXF plugin

To analyze stream data with HAWQ, you must first configure a PXF plugin so that PXF understands how to read DataLoader's proprietary format in Avro and translate it into database tuples, which can then be directly accessed by HAWQ.

1. Download the `dataloader-gpxf-plugin-2.0.4-bin.tar.gz` from the web site. If you have dataloader-service already installed, you can find it in `/usr/local/gphd/dataloader-2.0.4/plugins/gpxf`.
2. Untar `dataloader-gpxf-plugin-<version>.tar.gz` and put the extracted jar files in PXF's lib directory, which also contains PXF's jar files such as `gpxf-<version>.jar` or `pxf-<version>.jar`. Currently, there are only two jars in the tar ball:
   - dataloader-gpxf-plugin-<version>.jar
   - dataloader-common-<version>.jar.
3. Add these two jars into Hadoop's classpath. Just modify hadoop-env.sh and adding them to the list in $HADOOP_CLASSPATH, as PXF does.
4. Restart HDFS to implement the change.

You can now test HAWQ on DataLoader's stream data.

End-to-End Use Case

Currently, DataLoader's push stream CLI support two formats: TEXT and Avro. You can push either common text files or Avro data files to HDFS in a dataloader job.

To load the data into HAWQ, the data for a particular push stream job must have the same data schema across source clients connected to the job. You must create separate push stream jobs for streams with different schema.

Loading TEXT Format into HAWQ

For TEXT format, the file can be a CSV format where each line represents a record and each record contains fields separated by delimiters such as commas, for example:

```
John,male,18
Mary,female,30
David,male,70
```

The following example will use this sample text file.

1. First, push data from the text file, using the DataLoader CLI. In his example, we will use pcat.

```
$ dataloader-cli.sh pcat -n <job-id> --source sample_text_file --format TEXT
```

2. Invoke the HAWQ console with psql and create an external table, using the destination folder specified for the push stream job when the job is initiated.

```
gpadmin=# create external table csv_table (username text, gender text, age int)
location('pxf://namenode:50070/job/target/path/Job-Id?FRAGMENTER=HdfsData
Fragmenter&ACCESSOR=DataLoaderAvroFileAccessor&RESOLVER=TextResol
ver') FORMAT 'TEXT' (DELIMITER=',');
```

   The table is created with the following database schema:

Field Name	Data Type
username	text
gender	text
age	int

   The format is set as "TEXT" using commas as the delimiters, matching the data schema of the sample text files.

3. You can now make SQL queries using the HAWQ console.

```
gpadmin=# select username from csv_table;
```

Loading Avro Data into HAWQ

Avro format requires that all Avro format files have the same Avro schema. For example:

```
{
 "type" : "record",
 "name" : "Person",
 "doc" : "A sample Person record which contains fields like name, gender and age."
 "fields" : [
            {
              "name" : "name",
              "type" : "string",
              "doc" : "Name of the person"
            },
            {
              "name" : "gender",
              "type" : "string",
              "doc" : "Gender of the person"
            },
            {
              "name" : "age",
              "type" : "int",
              "doc" : "Age of the person"
            }
          ]
}
```

Suppose you create an Avro file named `sample_avro_file` using the above Avro schema, the following steps describe how to push the file into HAWQ:

1. First, push data from the Avro file using the DataLoader CLI. In this example, we will use pcat.

```
$ dataloader-cli.sh pcat -n <job-id> --source sample_avro_file --format AVRO
```

2. Invoke the HAWQ console with psql and create an external table, using the destination folder specified for the push stream job when the job is initiated.

```
gpadmin=# create external table avro_table (username text, gender text, age int)
location('pxf://namenode:50070/target/path/Job-Id?FRAGMENTER=HdfsDataFragmenter&ACCESSOR=DataLoaderAvroFileAccessor&RESOLVER=AvroResolver')
FORMAT 'custom' (formatter='pxfwritable_import');
```

The table is created with the following database schema, mtching the Avro schema in the sample file:

Field Name	Data Type
username	text
gender	text
age	int

3. You can now make SQL queries using the HAWQ console.

```
gpadmin=# select username from avro_table;
```

A. Data Stores

A data store is a abstraction for a persistent store that could be used to store data.. This appendix describes the properties used to register each supported data store.

Source and Destination Datastores

DataLoader has requirements for how the source data store is configured. These are detailed in the sections below.

Currently, only HDFS/HDFS2 is suported as a target datastore. Configuration is the same as for the source datastores.

FTP Data Store

The table below shows the properties of each type of FTP data store.

Table 1 FTP Data Store Registration Properties

Property	Values
type	ftp
host	The name or IP address of the FTP server.
rootPath	The root path of the source FTP server.
port	The port of the FTP server. The default port is 21.
user	The FTP username. Set as anonymous, if no authentication is needed.
passwd	The FTP user's password.Set as password, if no password is needed.
transfermode	The data transfer mode for this FTP server. Values can be "stream," "block," or "compressed." Default is "stream."
passive	When the FTP server is configured as passive, this value should be set to true.
timeout	The timeout value used to connect to the FTP server.
filetype	The type of the files on this FTP server. Can be "binary" or "ascii". Default is binary.

If using the Web GUI, use the following values for the FTP data stores:

Type: ftp
Host: <hostname>
Port: 21
Rootpath: /
User: ftpuser
Password: ftpuser

LocalFS Data Store

Using LocalFS as the source datastore has restrictions.

Data must be put in a directory that is universally readable, such as /data or /tmp, so that the mapreduce user can access it to copy the data.

LocalFS needs to be on local disk of a DataLoader worker (Hadoop Datanode) for distributed mode, on the manager node for standalone and pseudo-distributed mode.

To register LocalFS datastores, the slave machines must be used. You can register any DataLoader worker machines as LocalFS type data stores.

When using DataLoader in distributed mode, for LocalFS data stores, you must change the Fairscheduler with DataLoader's modified scheduler. The cluster must be restarted for HDFS 2.x.

The LocalFS host must be within the mrv1 or mrv2 cluster.

Refer to the table below for the Register Property values.

Table 2 LocalFS data store register properties

Property	Values
type	localfs
host	The name of a local host machine. Data is copied from this machine when you select the localfs strategy.
rootPath	The base path of the file system.
Username	User who has access to LocalFS
password	User's Linux password
port	Port of the ssh connection. Default is 22.

If using the Web GUI, use the following values for the LocalFS data stores:

Type: localfs
Host: <hostname>
Rootpath: /
User: test_user
Password: test_user

If using the CLI, use the following entries in the Property file:

```
type=localfs
host=<hostname>
rootpath=/
username=test_user
password=test_user
```

Configuring Hadoop for Loading Data from Local Filesystem

To copy data from the Local Filesystem, you will need to configure your Hadoop cluster to use our specialized FairScheduler.

FairScheduler setup

If you want to transfer data from the local file system, use the localfs copy strategy. With localfs copy strategy, DataLoader assigns the copy task to the node that holds the data, allowing the copy task to read data directly from the local filesystem. FairScheduler has been modified to enforce copy tasks being assigned locally for localfs copy. Set up the FairScheduler according to the following procedure.

YARN/MRV2:

1. If you are using YARN/MRv2 Hadoop cluster, copy the `dataloader-fairscheduler-mrv2*.jar` file to `$YARN_HOME/lib`, e.g. /usr/lib/gphd/hadoop-yarn/lib on the yarn resource manager. You can find this file in /usr/lib/gphd/dataloader-{version}/plugins/fairscheduler.
2. Configure YARN to use dataloader-fairscheduler as Resource Manager's scheduler. You can do this by setting a property in yarn-site.xml on the resource manager..

```
<property>
  <name>yarn.resourcemanager.scheduler.class</name>
<value>com.pivotal.hd.yarn.server.resourcemanager.scheduler.fair.FairScheduler</value>
</property>
```

3. If the Resource Manager is running, restart it for the changes to take effect.

MRV1:

1. If you are using the MRv1 Hadoop cluster, replace the Hadoop Fairscheduler jar file under `$HADOOP_HOME/lib/hadoop-fairscheduler*.jar` with `dataloader-fairscheduler-mrv1*.jar`.
2. Configure JobTracker to use dataloader-fairscheduler by setting a property in mapred-site.xml as shown below.

```
<property>
 <name>mapred.jobtracker.taskScheduler</name>
 <value>org.apache.hadoop.mapred.FairScheduler</value>
</property>
```

3. If JobTracker is running, restart it for the changes to take effect.

You can now load data from the localfs.

Impact to Hadoop environment

Setting FairScheduler to the specialized version can have an impact on the Hadoop environment. By setting up use of the specialized version of FairScheduler for DataLoader, the Hadoop cluster still works as if you set it up to use the original FairScheduler.

NFS Data Store

Using NFS as a source datastore has the following restriction:

NFS share must be mounted on all the nodes, including the master and workers with the same directory structure.

NFS Data Store Registration Properties:

Property	Values
type	nfs
host	The host of the NFS share.
rootPath	The starting path relative to the NFS mount point.
port	Port of the nfs share.
mountPoint	The NFS mount point on DataLoader master and worker nodes of this NFS share.

If using the Web GUI, use the following values for the NFS data stores:

Type: nfs
Host: <hdfs namenode host>
Rootpath: /
Mountpoint: /mnt/nfs

If using the CLI, use the following entries in the Property file:

```
type=nfs
host=<hostname>
rootPath=<rootpath>
mountPoint=<mount_point>
```

The nfs rootPath represents the share path of the current nfs share.

HDFS Data Store

HDFS datastore is able to connect to HDFS that comply with Apache Hadoop 1.x.

HDFS authentication should be disabled if using it as a source datastore.

The table below shows the properties of each type of HDFS data store.

HDFS Data Store Properties

Property	Values
type	hdfs
host	The namenode server of the HDFS cluster.
port	The namenode port of the HDFS cluster.
rootPath	The starting/base of the HDFS namespace.

HDFS values if using Web GUI:

Type: hdfs

Host: <hdfs namenode host>

Port: <hdfs port>

Rootpath: =/

If using the CLI, use the following entries in the Property file:

```
type=hdfs
host=<hostname>
rootPath=<rootpath>
port=<port>
```

HDFS2 Datastore

HDFS2 datastore is used to connect to HDFS that comply with Apache Hadoop 2.0.2. This type of datastore has the same restrictions as the HDFS datastore. .

Table A.1 HDFS2 data store registration properties

Property	Values
type	hdfs2
host	The namenode server of the HDFS cluster.
port	The namenode port of the HDFS cluster. The Pivotal HD default port is 8020.
rootPath	The starting/base of the HDFS namespace.

Values if using Web GUI:

Type: hdfs2

Host: <hdfs namenode host>

Port: <hdfs port>

Rootpath: =/

If using the CLI, use the following entries in the Property file:

```
type=hdfs2
host=<hostname>
rootPath=<rootpath>
port=<port>
```

How set up an NFS server and client

Set up the NFS server

Run the following command to install NFS:

```
# yum install nfs-utils nfs-utils-lib
```

Run the following commands to start NFS:

```
# cd /etc/init.d
# ./rpcbind start
# ./nfs start
```

Create the NFS directory on the local machine:

```
# mkdir /mnt
# mkdir /mnt/nfs
```

Edit the /etc/exports file and enter the paths to be accessed by all clients.

Note: Use hdsh* for all clients whose hostname starts with hdsh.

Open the /etc/exports file for editing:

```
# vi /etc/exports
```

Enter the following content in the /etc/exports file:

```
/mnt/nfs hdsh*(rw,all_squash,sync,fsid=0)
```

Exit the file and run the command:

```
# exportfs -ra
```

Set up the NFS Client

You must install the NFS Client on all DataLoader worker nodes.

1. To install the NFS client, enter:

```
# yum install nfs-utils nfs-utils-lib
```

2. Start NFS:

```
# cd /etc/init.d
# ./rpcbind start
# ./nfs start
```

3. Create the NFS directory:

```
# mkdir /mnt
# mkdir /mnt/nfs
```

4. Mount the NFS directory.

Note: NFS should be mounted on all the nodes (master, workers) on the same path.

   a. Mount the client directory to the NFS server directory.

```
mount -t nfs <nfs-server>:<server directory to be shared with all
nfs clients> <client directory to be mounted>
```

   b. Execute the following command on all DataLoader nodes:

```
# mount -t nfs <nfs-server>:/mnt/nfs /mnt/nfs
```

How to set up the FTP server and client

1. Install vsftpd:

```
# yum install vsftpd
# chkconfig vsftpd on
# service vsftpd start
# service iptables start
```

2. Add iptables entry for port 21.

a. Edit the iptables.

```
# vi /etc/sysconfig/iptables
```

b. Add an entry in /etc/sysconfig/iptables for port 21 as shown below.

Note: The entry should be added before rejecting related lines.

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 21 -j
ACCEPT
```

a. Close the firewall on the server.

```
# /etc/init.d/iptables stop
```

3. Install FTP.

```
# yum install ftp
```

4. Add the ftp user on the Linux machine:

```
# useradd ftpuser
# passwd ftpuser
```

5. Add the same user (ftpuser) for ftp access. Open /etc/vsftpd/chroot_list for editing:

```
vi /etc/vsftpd/chroot_list
```

6. Add the ftp user name ftp user to the file:

```
[root@centos-1 \~\]# cat /etc/vsftpd/chroot_list
 ftpuser
```

7. Restart the service.

```
# service vsftpd restart
```

8. Install the ftp Client.

```
yum install ftp
```

Note: Because ftpuser was used to set up ftp service, the root folder is at /home/ftpuser.

The ftp Client must be installed on all DataLoader Worker nodes. Files must be created in the /home/ftpuser directory on the ftp server for DataLoader to find and upload them.

Datastore Configuration Examples

The following datastore examples have been tested.

HDFS v1 configuration

```
type=hdfs
host=hdsh070
port=54310
rootPath=/
```

HDFS v2 configuration

```
type=hdfs2
host=hdsh070
port=8020
rootPath=/
```

NFS setup and configuration

```
type=nfs
host=hdsh070
rootPath=/
mountPoint=/mnt/nfs
```

FTP server setup/configuration

```
type=ftp
host=hdsh070
port=21
rootPath=/pub
username=anonymous
password=guest
```

Local FS environment setup

```
type=localfs
host=hdsh076
rootpath=/
username=test_user
password=test_user
```

B. Job Specification

What is a Job Specification (JobSpec)

A Job Specification or JobSpec (also known as a Transfer Spec) is a hierarchal XML-based file that defines a job being submitted. It is called by the `dataloader-cli.sh submit -i <transfer specification>` command. Refer to the "File Spec Samples" on page 53 for examples of the use of the elements and attributes used in these files.

The "JobSpec Elements and Attributes" table below shows the elements and attributes of each of these types.

Table B.1 JobSpec Elements and Attributes

element	Element/attribute	Attribute/Value
name: The name is the name of the job.		
type: The **type** can be either **batch** or **streaming**.	**batch** or **streaming**.	
streaming: if the type is "streaming", this element should be set	**type**: this is an attribute of the streaming element	available values are: **push**, **pull**
	`feedSet`: this element specifies the content to be copied; each feedSet includes data in one datastore; this element can occur many times.	**`datastore`**: type URI, specifies the datastore; can occur only once. **`feed`**: can occur multiple times. It can have only one attribute: path, which specifies the actual path to be copied.
	`destination`: this element can occur only once. It specifies the destination of the streaming job datastore: this element is of type URI, specifies the target datastore; can occur only once	
	`path`: string element. Specifies the location on the target datastore where data is to be copied. In Pivotal HD 1.0, can be used only once.	
	`reader`: element of type "ReaderType". Specifies the reader that will be used to read data from the source datastore. Can occur only once.	`property`: a key-value pair used as the reader properties. Can occur multiple times.

Table B.1 JobSpec Elements and Attributes

element	Element/attribute	Attribute/Value
	pipeline: this element specifies the pipeline that will be used	worker: this attribute specifies the worker that will be used in the pipeline; can occur many times; name: the name of the worker; order: int value, the order of this worker in the pipeline; value should not be the same
	writer: this element is of type "WriterType", specifies the writer that will be used to write data to the target datastore; can occur only once	**property**: a key-value pair used as the reader properties. Can occur multiple times.
batch: if the type is batch, this element should be set	**fileSet**: this element specifies the files/items to be copied; each fileSet includes data in one datastore; this element can occur many times	**datastore**: URI value, specifies the source datastore; can occur only once; **filePath**: this element is of type "FilePathType", which specifies the path that needs to be copied -- can occur multiple times **globString**: glob string used to filter the path, only valid if the type if "glob" **filePath**: recursive value of the same type "FilePathType", provides more detailed file listing functions **filter**: attribute, specifies the file name filter **type**: attribute, the type of this path, available values are: "file", "folder", "glob", "list" **path**: attribute, the exact file path that this filePath specifies **disk**: attribute, the disk that this filePath resides on, only valid if the datastore is localfs
	destination: the destination of the data; can occur only once	**datastore**: the URI of the target datastore path: this attribute is of type "FilePathType", specifies the target location; this one should only use "path" attribute; trimPrefix: the name mapping of the source file to destination file; this value will be trimmed from the start of the source path
	reader: element of type "ReaderType". Specifies the reader that will be used to read data from the source datastore. Can occur only once.	**property**: a key-value pair used as the reader properties. Can occur multiple times.

Table B.1 JobSpec Elements and Attributes

element	Element/attribute	Attribute/Value
	pipeline: this element specifies the pipeline that will be used	**worker**: this attribute specifies the worker that will be used in the pipeline; can occur many times; • name: the name of the worker; • order: int value, the order of this worker in the pipeline; value should not be the same
	writer: this element is of type "WriterType", specifies the writer that will be used to write data to the target datastore; can occur only once	**property**: a key-value pair used as the reader properties. Can occur multiple times.

File Spec Samples

Sample ftp.xml file with file entry type "glob":

```
<JobSpec xmlns="http://www.pivotal.com/hd/dataloader">
<name>batch-job-1</name>
<type>batch</type>
<batch>
<fileSet>
<datastore>ftp://YOUR_SOURCE_HOSTNAME:21/</datastore>
<filePath type="glob" path="/" >
<globString>/movie/**/*.MOV</globString>
</filePath>
</fileSet>
<destination>
<datastore>hdfs://TARGET_HDFS_NAMENODE:8020/data</datastore>
<path type="folder" path="/2013/01"></path>
<trimPrefix>/movie</trimPrefix>
</destination>
<reader type="FS_READER">
</reader>
<writer type="FS_WRITER">
</writer>
</batch>
</JobSpec>
```

Note: For the ftp.xml transfers, filenames that end with ".MOV" from the source `ftp://<YOUR_SOURCE_HOSTNAME:21/` maintain the folder structure, but the "/movie" is trimed off from the folder hierarchy. Sub-folders of the movie folder will be copied to the destination HDFS hdfs://<TARGET_HDFS_NAMENODE>:8020/data/2013/01

Sample localfs.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<JobSpec xmlns="http://www.pivotal.com/hd/dataloader">
<name>localfs job</name>
<type>batch</type>
<batch>
<fileSet>
<datastore>localfs://YOUR_SLAVE1_HOSTNAME/datastore_rootpath<
/datastore>
<filePath type="folder" path="/logs" filter="*.log">
</filePath>
</fileSet>
<fileSet>
<datastore>localfs://YOUR_SLAVE2_HOSTNAME/datastore_rootpath<
/datastore>
<filePath type="folder" path="/logs" filter="*.log">
</filePath>
</fileSet>
<destination>
<datastore>hdfs://YOUR_TARGET_NAMENODE:8020/data</datastore>
<path type="folder" path="/2013/01"></path>
<trimPrefix>/logs</trimPrefix>
</destination>
<reader type="FS_READER">
</reader>
<writer type="FS_WRITER">
</writer>
</batch>
</JobSpec>
```

This spec will transfer the files in the folder "/logs" on the two specified localfs datastores to the target HDFS.

Sample localfs_disk.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<JobSpec xmlns="http://www.pivotal.com/hd/dataloader">
<name>localfs job</name>
<type>batch</type>
<batch>
<fileSet>
<datastore>localfs://YOUR_SLAVE1_HOSTNAME/datastore_rootpath</dat
astore>
<filePath type="folder" path="/logs/data1" filter="*.log"
disk="disk1">
</filePath>
<filePath type="folder" path="/logs/data2" filter="*.log"
disk="disk2">
</filePath>
<filePath type="folder" path="/logs/data3" filter="*.log"
disk="disk3">
</filePath>
</fileSet>
<fileSet>
<datastore>localfs://YOUR_SLAVE2_HOSTNAME/datastore_rootpath</dat
astore>
<filePath type="folder" path="/logs" filter="*.log" disk="disk1">
</filePath>
</fileSet>
<destination>
<datastore>hdfs://YOUR_TARGET_NAMENODE:8020/data</datastore>
<path type="folder" path="/2013/01"></path>
<trimPrefix>/logs</trimPrefix>
</destination>
<reader type="FS_READER">
</reader>
<writer type="FS_WRITER">
</writer>
</batch>
</JobSpec>
```

This specification specifies the similar data to be transferred with the data specified above, but with the disk information added. This information is useful when using localfs strategy with WORKERS_PER_DISK set in the transfer policy. See Appendix

F, "DataLoader Command Line Reference" for more detail. In the sample, "YOUR_SLAVE_HOSTNAME" is the hostname of a slave machine. Note that multiple disks can reside on the same host, but must be different physical disks.

Sample nfs.xml file with file entry type "folder"

```
<?xml version="1.0" encoding="UTF-8"?>
<JobSpec xmlns="http://www.pivotal.com/hd/dataloader">
<name>nfs job</name>
<type>batch</type>
<batch>
<fileSet>
<datastore>nfs://NFS_SERVER_HOST/datastore_rootpath</datastore>
<filePath type="folder" path="/logs" filter="*.log"></filePath>
<filePath type="folder" path="/logs2" filter="*.log"></filePath>
</fileSet>
<destination>
<datastore>hdfs://YOUR_TARGET_NAMENODE:8020/data</datastore>
<path type="folder" path="/2013/01"></path>
<trimPrefix>/logs</trimPrefix>
</destination>
<reader type="FS_READER">
</reader>
<writer type="FS_WRITER">
</writer>
</batch>
</JobSpec>
```

The specification file transfers folders /logs and /log2 from the source `nfs://NFS_SERVER_HOST/datastore_rootpath` to the destination `hdfs://YOUR_TARGET_NAMENODE:8020/data`.

Sample hdfs.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<JobSpec xmlns="http://www.pivotal.com/hd/dataloader">
<name>hdfs job</name>
<type>batch</type>
<batch>
<fileSet>
<datastore>hdfs://YOUR_SOURCE_NAMENODE:8020/datastore_rootpath</d
atastore>
<filePath type="folder" path="/logs" filter="*.log">
</filePath>
</fileSet>
<destination>
<datastore>hdfs://YOUR_TARGET_NAMENODE:8020/data</datastore>
<path type="folder" path="/2013/01"></path>
<trimPrefix>/logs</trimPrefix>
</destination>
<reader type="FS_READER">
</reader>
<writer type="FS_WRITER">
</writer>
</batch>
</JobSpec>
```

Sample Pushstream job

```
<JobSpec xmlns="http://www.pivotal.com/hd/dataloader">
<name>name</name>
<type>streaming</type>
<streaming type="push">
<destination>
<datastore>hdfs://YOUR_TARGET_NAMENODE:8020/</datastore>
<path>/tmp</path>
</destination>
<reader type="PUSH_STREAM_READER">
</reader>
<writer type="PUSH_STREAM_WRITER">
                <!-- rolling.size is 10KB -->
      <property name="rolling.size"    value="10240"/>
</writer>
</streaming>
</JobSpec>
```

This spec is telling DataLoader to create a pushstream job; the received data will be stored to the HDFS datastore `hdfs://YOUR_TARGET_NAMENODE:8020/`, into the folder `/tmp`. The number of workers to start is specified in the transfer policy.

Rolling size should be set by default to 64MB, the HDFS block size, for maximum performance. For small streams or for demonstration purposes, set to a small number such as 10KB (10240).

C. DataLoader Copy Strategies

Copy Strategies

DataLoader supports 6 copy strategies, shown in the table below.

Table C.1 DataLoader Copy Strategies

Strategy Name	Description	Supported Source	Supported Target
locality	Locality Copy Strategy exploits the location information from the data source, such that the job splits created by Job Planner will be assigned to workers having the highest possibility being co-located with the data blocks, thus reducing inter-machine traffic. Locality will be achieved during reads when DataLoader is deployed on source HDFS cluster, and during writes when deployed on destination HDFS cluster.	HDFS	HDFS/HDFS2 supports concat*
localfs	LocalFS strategy is a special case of Locality strategy where source data is on the local disks of the DataLoader worker nodes. The location information must be used to verify that the workers are copying from the disk mounted on the same server, as the local file system can not be accessed from a remote machine via DataLoader. LocalFS strategy tells the DataLoader scheduler to assign the file splits only to the host where the files are stored. LocalFS strategy supports additional optimization on top of host level assignment. In transfer specification files, users can identify the disk where the files are stored, and can specify the number of worker processes to copy from one disk. DataLoader scheduler will schedule the specified number of worker processes per disk to maximize usage of the available disk bandwidth available. Localfs strategy is NOT supported with YARN/MapReduce2	Raw file system	HDFS, HDFS2
uniform	Uniform strategy uniformly assigns loading tasks to all the loader machines according to file size.	FTP, NFS	HDFS, HDFS2

Table C.1 DataLoader Copy Strategies

Strategy Name	Description	Supported Source	Supported Target
Dynamic	Dynamic strategy is a batch copy strategy where the copy tasks are assigned on demand rather than pre-allocated in batch mode, i.e., rather than generating the job spits beforehand, the copy tasks are pooled and allocated to workers at run time. This strategy is used when a large number of small files must be copied. In a heterogeneous environment, it is easy to see long tail effects if the tasks are assigned more or less uniformly. The benefits of using dynamic strategy is that, the workers will be kept busy as long as there are still tasks to work on, whereas if a static allocation is used, some workers will finish and exit early and leave the stragglers running till complete. The downside of using dynamic strategy is, when there are large volume of data needs to be copied, usually the number of workers will scale up, and they will compete for the tasks to run, which inevitably causes contention and eventually hurts scalability.	NFS, FTP, HDFS, HDFS2	HDFS, HDFS2
connection limited	This limits the connections to data sources by the specified number of workers. This strategy is used when users are loading from FTP, or other data sources where the connections need to be throttled per the configured server capacity.	FTP	HDFS, HDFS2
pushstream	Pushstream copy strategy is used when a user submits a pushstream job. This strategy will allow the user to specify the number of workers to support the client it is connecting to. Similar to pull streaming job, Data Loader scheduler will over provision the number of required workers to support the job to make sure the low latency scheduling can be achieved.	N/A	HDFS, HDFS2
intelligent	With this strategy, DataLoader will automatically pick the suitable copy strategy for the user scenario. For example, if copying from HDFS, and target HDFS supports concat, then locality strategy will be selected; if copied from local file system, localfs strategy will be selected; otherwise uniform strategy will be used.	All data source	HDFS, HDFS2

Copy Strategies for Intelligent Copy

Table C.2 Strategies for Intelligent Copy

Source Datastore	Destination Datastore	Intelligent Approach
HDFS	HDFS with concat support	locality strategy
HDFS	HDFS without concat support	uniform strategy
Local FS	HDFS with concat support	localfs with chunking
Local FS	HDFS without concat support	localfs
NFS	HDFS with concat support	uniform strategy with chunking
NFS	HDFS without concat support	uniform strategy
FTP	\	uniform strategy

In Intelligent Copy, the DataLoader determines the best strategy based on the source and target data store configuration.

D. Job Transfer Policy

Transfer policy

A transfer policy is a set of user-specified configurations that impact the behavior of a particular job in DataLoader. When submitting jobs from the command line, the options are defined in a plaintext file and the file name is specified on command line.

When submitting jobs from Web UI, the options are specified in a form

Table D.1 Transfer Policy Options

Configuration Options	Description	Other Comments
WORKER_NUMBER	The number of workers to be started for a job. **NOTE**: This configuration option is only a hint for data loader. The number of workers actually started might not accurately equals to the figure user has specified. It depends on the WORKER_NUMBER option, the workload (e.g. a set of files) which a job works on and the copy strategy user has selected.	
STRATEGY	The copy strategy user has select for the job. Refer to the "copy strategy" section for detailed explanations.	"intelligent" copy strategy will select the best-fit-in strategy for a job.
IS_OVERWRITE	If a file with the same name already exists in destination, should it be overwritten or not by the file to be transferred. This setting is not applicable to streaming jobs.	
USE_COMPRESSION	Whether the payload transferred should be compressed when transferred to destination. Currently Google's Snappy codec is used for compression.	"USE_COMPRESSION" and "IS_CHUNKING" CANNOT be enabled together
IS_CHUNKING	Whether the payload should be split into small pieces and transferred separately to destination. When all the parts are transferred to destination HDFS successfully, they will stitched together to restore back the original payload. **NOTE**: To do chunking, it requires the destination hdfs support concat() operation to stitch the small pieces together.	"USE_COMPRESSION" and "IS_CHUNKING" CANNOT be enabled together

Table D.1 Transfer Policy Options

Configuration Options	Description	Other Comments
CHUNKING_SIZE	The size of a chunk when IS_CHUNKING is enabled.	Only valid if IS_CHUNKING is enabled
CHUNKING_THRESHHOLD	Only when the size of the file exceeds the value of CHUNKING_THRESHOLD, will it be chunked	Only valid if IS_CHUNKING is enabled
IS_BAND_WIDTH_THROTTLING	Whether to enable bandwidth throttling during the data transfer. When enabled, the max bandwidth that a worker can be used is specified by "BAND_WIDTH" option. Defaults to false.	
BAND_WIDTH	The max bandwidth which could be consumed by the started workers in a job. This option tends to control the resource consumption in a share cluster. The unit for this option is "bytes/s".	Only valid if IS_BAND_WIDTH_THROTTLING is enabled.
WORKERS_PER_DISK	The number of workers that will copy the files in a single disk.	
EVENTS_FLUSH_BUFSIZE	When the size of the DataLoaderWorker memory buffer that holds events from the client exceeds EVENTS_FLUSH_BUFSIZE, DataLoaderWorker will flush all events in its buffer to persistent storage for checkpointing. Increasing EVENTS_FLUSH_BUFSIZE appropriately could enhance the I/O throughput of DataLoader. Alternatively, it might usemore system resources, as the DataLoader worker would need more memory resources to hold the events coming from the client. Setting EVENTS_FLUSH_BUFSIZE to a small value could enhance event latency for events but hamper I/O throughput.	Valid only for pushstream jobs. When no value is specified, this parameter is inactive.

Table D.1 Transfer Policy Options

Configuration Options	Description	Other Comments
EVENTS_FLUSH_NUM	When the number of events in DataLoaderWorker's memory buffer exceeds EVENTS_FLUSH_NUM, DataLoaderWorker will flush all events in its buffer to persistent storage for checkpointing. Increasing EVENTS_FLUSH_NUM appropriately could enhance the I/O throughput of DataLoader. Alternatively, it might usemore system resources, as the DataLoader worker would need more memory resources to hold the events coming from the client. Setting EVENTS_FLUSH_NUM to a small value could enhance event latency for events but hamper I/O throughput.	Valid only for pushstream jobs. When no value is specified, this parameter is inactive.
EVENTS_FLUSH_INTERVAL	The interval at which DataLoaderWorker will flush all events in memory buffer to persistent storage for checkpointing (in miliseconds). Default value: 5000ms Increasing EVENTS_FLUSH_INTERVAL appropriately could enhance the I/O throughput of DataLoader. Setting EVENTS_FLUSH_INTERVAL to a small value could enhance event latency for events but hamper I/O throughput.	Valid only for pushstream jobs.
STREAM_DATA_DURABILITY	Whether event data durability should be enabled for this pushstream job.	Not defined if this is a batch job.

.

E. Supported Combinations of Job Copy Strategy and Transfer Policy

When loading from different data sources, the user must define the job spec and transfer policy before submitting a job for Data Loader to run.

See Appendix C, "DataLoader Copy Strategies" for more information on the copy strategies.

See Appendix B, "Job Specification" for information about elments of a Job Specification file, as well as samples.

See Appendix D, "Job Transfer Policy" for information on the transfer policy options.

The following Table defines the applicable settings and how they can be combined to create a job.

Source Data Store Type	Supported Copy Strategy	Supported Policy	File Transfer Spec
HDFS	locality uniform connection limited dynamic intelligent*	Chunking (If destination data store support concat) Bandwidth-throttling Overwrite worker-number Compression (chunking needs to be disabled)	folder file glob list
NFS	uniform connection limited dynamic intelligent*	Chunking (if destination cluster support concat) Bandwidth-throttling Overwrite worker-number Compression (chunking needs to be disabled)	folder file glob list
LocalFS	localfs intelligent*	Chunking Bandwidth-throttling Overwrite worker-number worker-per-disk Compression (chunking needs to be disabled)	folder file glob list
FTP	uniform connection limited dynamic intelligent*	Bandwidth-throttling Overwrite worker-number Compression (chunking needs to be disabled)	folder file glob list

Note: By using *intelligent* strategy, DataLoader will choose the most suitable strategy tor performing the copy.

Note: By using *intelligent* strategy, DataLoader will choose one suitable strategy to perform the copy. In this case, only *httppull* can/will be selected.

Source Data Store Type	Supported Copy Strategy	Supported Policy	File Transfer Spec
N/A In push stream scenarios, data loader accepts events/messages from any active client.	pushstream intelligent*	worker-number Compression Bandwidth-throttling	Transfer spec needs to specify the root path in its destination fs. All msgs/events DataLoader has received will be serialized in files under the specified root path, which must follow certain naming conventions.

F. DataLoader Command Line Reference

Directory Structure

The files for the DataLoader Command Line Reference are installed in the same folder as DataLoader service. By default the base directory is `/usr/local/gphd/dataloader-2.0.4`.

This package has no dependency on other DataLoader components.

The structure of this directory is as follows:

```
${dataloader_HOME}/
      bin/
      conf/
      lib/cli
      lib/cli/lib
```

The *bin* folder contains the dataloader-cli.sh script.

The *conf* folder contains the configuration files for this component.

The lib/cli folder contains the dataloader-cli**.jar and the folder lib/cli/lib contains the other required libraries.

DataLoader Commands

The client utility, dataloader-client.sh, supports multiple commands and associated options.

`Command Format:`

```
dataloader-cli.sh [COMMAND] [OPTIONS]
```

The DataLoader utility supports the following commands:

- `submit`: used in submiting DataLoader jobs
- `suspend`: used to suspend/resume, cancel, or query a job
- `config`: configures DataLoader instances
- `pstream`: pushes data from STDIN to Dataloader pushstream job
- `pcat`: pushes data from the specified file to Dataloader pushstream job
- `ptail`: pushes data from the specified file to Dataloader pushstream job
- `scan`: pushes data from specified folder to Dataloader pushstream job

Note: tail, the command to push data from STDIN to Dataloader pushstream job, has been deprecated.

The following table provides a quick summary of the available CLI options.

CLI Command	Options	OptionDescription
submit	-b,--bandwidth <arg>	Maximum bandwidth each worker can use.
	-B,--is-bandwidth-throttle <arg>	Whether bandwidth throttling should be enabled: true or false.
	-c,--chunksize <arg>	Chunk size to split files into, if chunking is enabled.
	-D,--stream-data-durability <arg>	Whether checkpointing should be enabled for streaming data durability, default value is false.
	-i,--input	Path to the copy job spec file.
	-k,--chunking <arg>	Whether data chunking in enabled: true or false.
	-m,--mappernum <arg>	Number of mappers used to do the transfer. Deprecated, use --workernum instead.
	-m, --max-disk-mapper <arg>	Maximunm mappers that can be started for one disk. Deprecated. Use workers-per-disk instead.
	-N,--name <arg>	Job name
	-o,--overwrite <arg>	When specified, the target file will be overwritten if it already exists at the destination. Otherwise, file copy will be skipped.
	-p,--policyfile <arg>	Path to the configuration file that specifies the transfer policy.
	-s,--strategy <arg>	Copy strategy to be used: intelligent, localfs, locality, uniform, etc.
	-s, --sourcepath <arg>	Directory path at source
	-s, --sourceuri <arg>	URI of the source data store.
	-t,--chunkingthreshold <arg>	Minimum file size to chunk, if chunking is enabled.
	-t, --targetpath <arg>	Target copy directory path at destination.
	-t, --targeturi <arg>	URI of the target data store.
	-w,--workernum <arg>	Number of workers used to perform the copy.
	-w, --workers-per-disk <arg>	Maximum workers that can be started for one disk

CLI Command	Options	OptionDescription
	-z,--compress	When specified, compress data using Snappy codec.
suspend	-n,--jobid <arg>	ID of the job to suspend.
resume	-n,--jobid <arg>	ID of the job to resume.
cancel	-n,--jobid <arg>	ID of the job to cancel.
query	-n,--jobid <arg>	ID of the job to query.
list	--status <arg>	list jobs with a specified status
config	--command <arg> .	The command used for config, valid values are: register, delete, list.
	-n <arg>	The datastore id to be removed (if this is an delete command)
tail		This command is deprecated, please use pstream instead
pstream	--delimiter <arg>	The delimiter used for spliting inputs, available values: NEW_LINE TAB EMPTY_SPACE COMMA SEMICOLON COLON DOT
	--format <arg>	Input data format, valid values are: AVRO, TEXT
	-n,--jobid <arg>	ID of job to be operated on
	-q,--queue-size <arg>	Maximum number of messages that are allowed to be held at the client, default is 32768.
pcat Note: This command will not monitor the file change. It will terminate when it reaches the end of the file.	--delimiter <arg>	The delimiter used for spliting inputs, available values: NEW_LINE TAB EMPTY_SPACE COMMA SEMICOLON COLON DOT
	--format <arg>	Input data format, valid values are: AVRO, TEXT
	--source<arg>	Specifies input file for the ptail/pcat command, must be specified!

CLI Command	Options	OptionDescription
ptail	--delimiter <arg>	The delimiter used for spliting inputs, available values: NEW_LINE TAB EMPTY_SPACE COMMA SEMICOLON COLON DOT
	--format <arg>	Input data format, valid values are: AVRO, TEXT
	--idle <arg>	If the file has not be changed for the specified amount of time, the file is considered as not changing anymore. If not specified, the program keeps polling the file forever. The unit is in milliseconds.
	-n,--jobid <arg>	ID of the job to perform actions.
	-q,--queue-size <arg>	Maximum number of messages that are allowed to be held at the client, default is 32768.
	- source <arg>	Specifies input file for the ptail/pcat command, must be specified!
scan	--delimiter <arg>	The delimiter used for spliting inputs, available values: NEW_LINE TAB EMPTY_SPACE COMMA SEMICOLON COLON DOT
	--format <arg>	Input data format, valid values are: AVRO, TEXT
	--idle <arg>	If the file has not be changed for the specified amount of time, the file is considered as not changing anymore. If not specified, the program keeps polling the file forever. The unit is in milliseconds.
	-n,--jobid <arg>	ID of the job to perform actions.
	-q,--queue-size <arg>	Maximum number of messages that are allowed to be held at the client, default is 32768.

CLI Command	Options	OptionDescription
	--scanfolder <arg>	The folder to be scanned for push stream data, only used when scan command is used!
	-tailing	If specified, the files will be monitored for changes and the changes are automatically pushed to Dataloader.
	--thread-pool-size <arg>	The maximum number of concurrent threads that can be used for processing files in this folder. Default is 5

dataloader-cli.sh

This is the DataLoader client utility.

Synopsis

```
dataloader-cli.sh [COMMAND] [OPTIONS]
```

The DataLoader utility supports the following commands:

- submit
- suspend
- resume
- stop
- query
- list
- config
- scan
- pstream
- ptail
- pcat

Submit

You must create a specification file before using the `submit` command.

The DataLoader command requires the following options:

```
$ dataloader-cli.sh submit -i <transfer specification> [-s
<strategy>, -m <mapper number>, -b <bandwidth>, -k true|false,
-c <chunking size>, -t true|false, -o true|false, -z]
```

Refer to Submit options and descriptions below for a full description of the options.

Sample DataLoader command

```
dataloader-cli.sh submit -i myfileset.xml -k true -o true -c
512M -m 24 -b 2M -s intelligent
```

You can expect the following when you issue this command:

- To receive a job id.
- Receive an error in the console window if the job fails

.

Table F.1 Submit options and descriptions

Option Name	Value type	Default value	Description
-i (--input)	Path to the job specification file.	N/A	Required. Specifies the location of the job specification file on the local file system
-m (--mappernum)	integer	0.	Deprecated. Use -w instead.
-B (--is-bandwidth-throttle)	boolean	N/A	Indicates bandwidth throttling should be enabled
-b (--bandwidth)	Long value. For example, 3M is interpreted as 3 megabytes.	No bandwidth control.	Defines maximum bandwidth to use. Only valid if bandwidth throttling is enabled
-k (--chunking)	Boolean.	False	Indicates whether the data chunking is enabled.
-c (--chunksize)	Long value with suffix. For example, 3M is interpreted as 3 megabytes.	64M	The size of each chunk file, if chunking is enabled.
-t (--chunkingthreshold)	Long value with suffix. For example, 3M is interpreted as 3 megabytes.	1.6G	The minimum file size to chunk, if chunking is enabled.
-s (--strategy)	One of the six: • hdfslocality • uniform • localfs • localdisk • connectionlimited • intelligent	intelligent	See Appendix C, "DataLoader Copy Strategies" for more information.
-o (--overwrite)	Boolean	false	Indicates whether Dataloader should overwrite the file if the file to be transferred already exists in the target location. If not, Dataloader will skip that file.

Table F.1 Submit options and descriptions

Option Name	Value type	Default value	Description
`-z` (/)	No value	/	Enable data compression.
--max-disk-mapper	Integer	0	Deprecated. Use `--workers-per-disk` instead.
-p (--policyfile)	String	N/A	Specifies the file that contains transfer policy. Value will be overwritten by the options specified in the command line.
-w (--workernum)	int	0	Specifies the number of workers to be started for this job.
--workers-per-disk	int		Specifies the maximum number of workers that can be started for each disk specified in teh job specification file. Only valid if copy LocalFS files and the strategy is LocalFS.
-D(--stream-data-durability)	boolean	false	Indicates whether event data durablity should be enabled for this push stream job(not used for batch job).

Job Operation Options

Job operation means to suspend/resume/cancel/query a job.

Suspend

To suspend a DataLoader data transfer job, enter:

```
dataloader-cli.sh suspend -n <jobid>
```

You can expect the following when you issue this command:

- To receive an error if the specified job does not exist or is not running.
- Suspends the target job

The options for this operation are as follows:

Option Name	Value type	Default value	Description
-n (--jobid)	String.	N/A	This value is required. Contains ID of the job to suspend.

Resume

To resume a suspended or failed DataLoader data transfer job, enter.

```
dataloader-cli.sh resume -n <jobid>
```

You can expect the following when you issue this command:

- To resume the target job.
- To receive an error if the specified job does not exist or is in an unexpected state.

The options for this operation are as follows:

Option Name	Value type	Default value	Description
-n (--jobid)	String.	N/A	This value is required. Contains ID of the job to resume.

Stop

To stop a DataLoader data transfer job, enter:.

```
dataloader-cli.sh stop -n <jobid>
```

You can expect the following when you issue this command:

- To stop the target job. A stopped job cannot be resumed.
- To receive an error message if the specified job does not exist or if the job has already stopped.

Option Name	Value type	Default value	Description
-n (--jobid)	String.	N/A	This value is required. Contains ID of the job to stop.

Query

You can query the progress of a specified data transfer job.

```
dataloader-cli.sh query -n <jobid>
```

You can expect the following when you issue this command:

- To receive status for the specified job.
- If Map Reduce is running, to receive the progress of the transfer.
- To receive an error if the specified job does not exist.

The options for this operation are as follows:

Option Name	Value type	Default value	Description
-n (--jobid)	String.	N/A	This value is required. Contains ID of the job to query.
- status	enum(RUNNING, STARTED, SUSPENDED, COMPLETED, FAILED		Job Status

List

To list all the running and completed jobs, enter:.

```
dataloader-cli.sh list --status <options>
```

The options for this operation are as follows:

Option Name	Value type	Default value	Description
--status	One of the following: STARTED COMPLETED CANCELED SUSPENDED RUNNING	N/A	Job Status.

Config

To see a full list of registered datastores, use the command:

```
dataloader-cli.sh config --command list
```

To register a new datastore:

```
dataloader-cli.sh config --command register -i <property_field>
```

To remove an existing datastore:

```
dataloader-cli.sh config --command delete -n <datastore _ID>
```

The options for this operation are as follows:

Option Name	Value type	Default value	Description
--command	One of the following: • list • register • delete	N/A	Configures the data store.

Option Name	Value type	Default value	Description
-i (/)	String	N/A	This is required for the register option. The name of the property file.
-n (/)	String.	N/A	Required for the delete command.

pstream:

To send data from STDIN to a running pushstream job, enter:

```
dataloader-cli.sh pstream -n <jobid>
```

The options for this operation are as follows:

Option Name	Value type	Default value	Description
-n (--jobid)	String.	N/A	The push stream job id to push data to.
-q --queue-size <arg>	Integer	32768	The maximum number of unacknowledged messages that are allowed to be held on the client side.
--format	enum(TEXT, AVRO)	TEXT	The format of the input file/stream.
--delimiter	enum(NEW_LINE TAB EMPTY_SPACE COMMA SEMICOLON COLON DOT)	NEW_LINE	The delimiter used to split the input file. Only used when the "format" option is "TEXT".

scan

To send data from a folder to a running pushstream job enter:

```
dataloader-cli.sh scan -n <jobid> --scanfolder
<folder_to_scan>
```

The options for this operation are as follows:

Option Name	Value type	Default value	Description
-n (--jobid)	String.	N/A	The push stream job id to push data to.
--scanfolder	String	N/A	The folder to be scanned for data to push to the DataLoader job
-q,--queue-size <arg>	Integer	32768	The maximum number of unacknowledged messages that are allowed to be held on the client side.

Option Name	Value type	Default value	Description
--idle	long	N/A	If the file has not been changed for this period of time, the file is considered as finished. Only valid in ptail/scan(if tailing is enabled).
--tailing	boolean	false	Only valid in scan command. If specified, the files in the scan folder will be monitored for changes.
--thread-pool-size	integer	5	The maximum number of concurrent threads that can be used for processing files. Only valid in scan command.
--format	enum(TEXT, AVRO)	TEXT	The format of the input file/stream.
--delimiter	enum(NEW_LINE TAB EMPTY_SPACE COMMA SEMICOLON COLON DOT)	NEW_LINE	The delimiter used to split the input file. Only used when the "format" option is "TEXT".

pcat:

To send data from a folder to a running pushstream job enter:

```
dataloader-cli.sh pcat -n <jobid> --source <inputfile>
```

The options for this operation are as follows:

Option Name	Value type	Default value	Description
-n (--jobid)	String.	N/A	The push stream job id to push data to.
--format	enum(TEXT, AVRO)	TEXT	The format of the input file/stream.
--delimiter	enum(NEW_LINE TAB EMPTY_SPACE COMMA SEMICOLON COLON DOT)	NEW_LINE	The delimiter used to split the input file. Only used when the "format" option is "TEXT".
--source	String	N/A	The file to be processed. Only valid for pcat/ptail.

ptail:

To send data from a folder to a running pushstream job enter:

```
dataloader-cli.sh ptail -n <jobid> --source <inputfile>
```

The options for this operation are as follows:

Option Name	Value type	Default value	Description
-n (--jobid)	String.	N/A	The push stream job id to push data to.
--format	enum(TEXT, AVRO)	TEXT	The format of the input file/stream.
--delimiter	enum(NEW_LINE TAB EMPTY_SPACE COMMA SEMICOLON COLON DOT)	NEW_LINE	The delimiter used to split the input file. Only used when the "format" option is "TEXT".
--source	String	N/A	The file to be processed. Only valid for pcat/ptail.
--idle	long	N/A	If the file has not been changed for this period of time, the file is considered as finished. Only valid in ptail/scan(if tailing is enabled).

To register/unregister datastores

Option Name	Value Type	Default Value	Description
--command	enum(register,delete,list)	N/A	The command to run, either register a datastore or delete a datastore.
-i (--input)	String	N/A	The file that contains the datastore properties. Only used when command is "register".
-n	String	N/A	The ID of the datastore to be deleted. ID of the datastore can be found via the command list. Only used when command is "delete".

G. Configuring Flume for push streaming

DataLoader provides an out-of-the-box plug-in to integrate Apache Flume with DataLoader. This section gives the steps to configure Flume to push data to DataLoader

Configuring Flume

1. Get the Flume sink plugin

   Download the `dataloader-pushstream-flumesink-2.0.4-bin.tar.gz` from the web site. If you have dataloader-service already installed, find it in `/usr/local/gphd/dataloader-2.0.4/plugins/flumesink`.

2. Put the sink jar into the Flume class path.

   Untar the file and put all the jars into flume's lib directory:

```
$ tar -xf dataloader-pushstream-flumesink-2.0.4-bin.tar.gz

$ cp dataloader-pushstream-flumesink-2.0.4.jar lib/*.jar
$FLUME_HOME/lib
```

   It will alert you some files may be duplicated, just skip these files.

3. Configure Flume to use DataLoader push stream sink.

4. The following is an example of a Flume configuration file:

```
a1.sources = r1
a1.sinks = k1
a1.channels = c1
# Describe/configure the source
a1.sources.r1.type = netcat
a1.sources.r1.bind = localhost
a1.sources.r1.port = 44444
# Describe the sink
a1.sinks.k1.type =
com.pivotal.hd.dataloader.flume.sink.PushstreamSink
a1.sinks.k1.dataloader-job-id = <job id for dataloader push
stream job started>
a1.sinks.k1.zk-address =
zkServer1:port1,zkServer2:port2,zkServer3:port3
# Use a channel which buffers events in memory
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100
# Bind the source and sink to the channel
a1.sources.r1.channels = c1

a1.sinks.k1.channel = c1
```

Note in the above file, only the sink configuration is related to DataLoader. For other configurations on Flume, refer to the *Apache Flume User Guide*.

The parameters are as follows:

- type: Specifies the name of the class to use, it should be "com.pivotal.hd.dataloader.flume.sink.PushstreamSink"
- dataloader-job-id: Is the corresponding DataLoader the job id for the push stream job which has already been started to accept data from remote client. In this case, flume sink is also a client of the push stream job. Flume will push data to this job.
- zk-address: The address for the ZooKeeper server and port, as a comma-separated list.

5. Start Flume and use it to push data to DataLoader.

H. Installing and Configuring DataLoader from binaries

DataLoader can manage a dynamic pool of loader machines for fast ingestion of big data. DataLoader works with both Hadoop 1.x (MR1) and Hadoop 2.x (Yarn) and can leverage them as the resource manager and job scheduler.

Note: A properly-configured Hadoop cluster is required, and HDFS, MapReduce, and Zookeeper must be available for DataLoader to use. For Hadoop/Zookeeper deployment instructions using the Pivotal HD distribution, refer to the *Pivotal HD Installation and Administrator Guide*.

These instructions cover installation from thebinary tarball distribution. For configuration instructions, refer to Chapter 2, "Installing and Configuring DataLoader".

Packages

DataLoader is distributed as either RPM packages, or as a binary distribution. The installation procedures for the binary distribution are similar to the RPM installation. Use the installation procedure below for the binary distribution.

Binary Distribution:

DataLoader uses two binary packages that are deployed to the Client and Master, nodes, which can be the same.

Table H.1 Binary Distribution packages

Package Name	Description	Deployment Nodes
dataloader-cli-2.0.4.tar.gz	dataloader-cli-2.0.4.tar.gz provides essential files for setting up the DataLoader client	Client
dataloader-service-2.0.4.tar.gz	dataloader-cli-2.0.4.tar.gz provides essential files for setting up the DataLoader master	Master

Prerequisites

Installation of User Account

Two user accounts are needed for DataLoader: dladminfor installation and dataloader for running the service.

Create an Installation User Account

1. Create a dladmin account on the host machines where you will install DataLoader. This dladmin account will be used as a system administrator account for installing DataLoader.
2. Add dladmin to the sudoers list.

```
# vi /etc/sudoers
```

3. This will take you to a text editor with an `/etc/sudoers` file already opened. Add the following line:

```
dladmin ALL=(ALL) NOPASSWD: ALL
```

   and save the file.

4. Change to the dladmin user.

```
[root@centos62-2 ~] su - dladmin
```

5. Create a dataloader account on the master node and add it to the root group.

```
[dladmin@centos62-2 ~]$ sudo useradd dataloader
[dladmin@centos62-2 ~]$ sudo usermod -G root dataloader
```

Pivotal HD and Zookeeper Cluster Installation

To install Pivotal HD and the Zookeeper cluster, refer to the refer to the *Pivotal HD Enterprise Installation and Administrator Guide* for deployment instructions. Make sure you include the following in your HDFS configuration

For HDFS 1.x:

```
<configuration>
  <property>
    <name>dfs.support.append</name>
    <value>true</value>
  </property>
</configuration>
```

For HDFS 2.x:

```
<configuration>
   <property>
      <name>dfs.support.append</name>
      <value>true</value>
   </property>
</configuration>
```

Install the JDK

1. Install the JDK: Download and install the Oracle JDK1.6 (Java SE6 or JDK 6) from:

http://www.oracle.com/technetwork/java/javase/downloads/index.html

(http://www.oracle.com/technetwork/java/archive-139210.html)

2. After installing JDK, set the JAVA_HOME environment variable referring to where you installed JDK. On a typical Linux installation with Oracle JDK 1.6, the value of this variable should be `/usr/java/default/`.

```
export JAVA_HOME=/usr/java/default/
```

3. Add $JAVA_HOME/bin into your PATH environment variable. On a Linux platform with bash shell, add the following lines into the file ~/.bashrc:

```
export PATH=$JAVA_HOME/bin:$PATH
```

Download and extract the DataLoader files

Create a folder called /opt/dataloader to be used for DataLoader installation.

```
[dladmin@centos62-2 dladmin]$ sudo mkdir /opt/dataloader
[dladmin@centos62-2 opt]$ sudo chown dladmin:dladmin /opt/dataloader
```

Download and copy the PHDTools stack to /home/dladmin/ on the host where you want to install DataLoader. Make sure the Tarball has read permission for the user 'dladmin'. You will now extract the tar files.

Extract the tarred binaries as follows:

```
[dladmin@centos62-2 dladmin]$ wget
http://hdsh129.lss.emc.com/dist/artemis/latest/PHDTools-1.0.1-bin-xx.tar.gz
[dladmin@centos62-2 dladmin]$ tar -xzvf PHDTools-1.0.1-bin-xx.tar.gz
```

Note: xx in the Package Name is the package build number.

Copy the dataloader-service and dataloader-cli tar balls to /opt/dataloader.

```
[dladmin@centos62-2 dataloader]$ cp
/home/gpadmin/PHDTools-1.0.1-bin-18/dataloader/* /opt/dataloader
```

Untar the tar balls.

```
[dladmin@centos62-2 dataloader]$ sudo tar -xzvf dataloader-service-2.0.4.tar.gz
[dladmin@centos62-2 dataloader]$ sudo tar -xzvf dataloader-cli-2.0.4.tar.gz
```

The following code sample shows the structure of the directories and files that appear under /opt/dataloader

```
[dladmin@centos62-2 dataloader]$ ls -l
total 178956
drwxr-xr-x 2 dladmin dladmin      4096 Jul  2 05:59 bin
drwxr-xr-x 4 dladmin dladmin      4096 Jul  2 06:02 conf
drwxrwxr-x 3 dladmin dladmin      4096 Jul  2 06:03 data
-rw-r--r-- 1 dladmin dladmin  18726533 Jul  2 05:58 dataloader-cli-2.0.4.tar.gz
-rw-r--r-- 1 dladmin dladmin 164491899 Jul  2 05:58 dataloader-service-2.0.4.tar.gz
drwxr-xr-x 9 dladmin dladmin      4096 Jun 25 10:01 lib
drwxrwxr-x 2 dladmin dladmin      4096 Jul  2 06:03 log
drwxrwxr-x 2 dladmin dladmin      4096 Jul  2 06:03 pid
drwxr-xr-x 5 dladmin dladmin      4096 Jun 25 10:07 plugins
```

Change the ownership of the base folder so that its owner is dataloader.

```
[dladmin@centos62-2 dataloader] sudo chown -R dataloader:dataloader
/opt/dataloader
```

Configure DataLoader

Based on the installation directory you specified in dl-cluster.conf file and your PHD/ZK cluster configuration, modify this file to configure DataLoader, as shown:

```
[dladmin@centos62-2 dataloader] sudo vi /opt/dataloader/conf/dataloader-env.sh
```

The following configuration options must be modified to run DataLoader in distributed mode. If no default is given, the option must be specified.

Table H.2

Parameter	Function	Default value	Options
DATALOADER_ZK_QUORUM	List of Zookeeper nodes in comma separated format. This must be the same as used in Zookeeper cluster configuration, that is usually found in zk_cfg file	None. Must be specified.	N/A
DATALOADER_SERVICE_IP	IP address the services will bind to. "localhost" cannot be used	Must be specified except when running standalone mode, when CLI is installed on the service node.	N/A
EXECUTOR_TYPE	Hadoop cluster version that Data Loader is running on. Data Loader supports Hadoop 1.0 (GPHD 1.2) and Hadoop 2.0(PHD 1.x) . Other distribution, including Apache, are not guaranteed.	mrv2	mrv1/mrv
HADOOP_INSTALL	The Hadoop installation directory. DataLoader uses it to set the classpath. Make sure this value is set correctly.	$ HADOOP_INSTALL If this shell env variable is not exported, or the value is empty, the parameter must be specified manually.	N/A

Table H.2

Parameter	Function	Default value	Options
HADOOP_CONF_DIR	The Hadoop configuration directory .If you are using install_worker in the tarball to install the worker, this value must be set. Otherwise, setting this parameter is optional.	$HADOOP_CONF_DIR If this shell env variable is not exported, or the value is empty, the parameter must be specified manually.	N/A
DATALOADER_EXECUTOR_CAPACITY	The total number of slots (MRv1) or containers (MRv2) that are available to DataLoader. This value must be set for DataLoader to run correctly.	No default value, must be specified except when using Standalone mode.	
DATALOADER_DATA_DIR	The location where DataLoader data files are placed.	In tarball installation, this parameter is set to $[DATALOADER_HOME/data (/opt/dataloader/data)	
DATALOADER_LOG_DIR		In tarball installation, this parameter is set to $[DATALOADER_HOME/log /opt/dataloader/log)	

Install DataLoader in Standalone mode

If you are using the standalone version of DataLoader, use the following installation procedure. DataLoader supports standalone mode without needing an existing Hadoop cluster.

For Standalone mode, you can run the installation commands as user "dladmin" or any user that has root privileges:

```
[dladmin@hdp2-w17 ~]./install_dl.sh -s
```

Data Loader service and CLI tool will be installed at the default location /`usr/local/gphd/dataloader-2.0.4/`.

After installation, no further configuration is needed.

Install DataLoader in Pseudo-Distributed Mode

A Hadoop 1.x or 2.x cluster should be installed for pseudo-distributed mode.

1. Run the following command as dladmin or as any user who has root privileges.

```
[dladmin@hdp2-w17 ~]./install_dl.sh -s
```

2. Data Loader service and the CLI tool will be installed at the default location /usr/local/gphd/dataloader-2.0.4/

Note: The installation command is the same as in Standalone Mode, but there are configuration steps to be performed for Pseudo-distributed mode before you start DataLoader processes.

Install DataLoader in Distributed Mode

Before starting installation, verify that the Hadoop cluster is running and that the Hadoop client is installed and configured on the installing host.

1. Modify the template configuration file to create a new configuration file.

```
[dladmin@hdp2-w17 ~]$ cp dl-cluster.conf.template dl-cluster.conf
```

2. If you need DataLoader to use a specific directory structure, modify the dl-cluster.conf file. Otherwise, use the default file contents. The sample file below shows the default contents of dl-cluster.conf.

```
#DataLoader home directory
HOME_DIRECTORY=/usr/local/gphd
#DataLoader log directory
LOG_DIRECTORY=/var/log/gphd
#DataLoader data directory
DATA_DIRECTORY=/var/lib/gphd
```

3. As dladmin, run the following command to install DataLoader on the current node:

```
[dladmin@hdp2-w17 ~]$ ./install_dl.sh -d -f dl-cluster.conf
```

Uninstalling DataLoader

There are two uninstall options. One preserves customer data, and the other uninstalls both DataLoader and customer data.

To uninstall DataLoader but preserve customer data, use the command:

```
[dladmin@hdp2-w17 ~]./install_dl.sh -u
```

To uninstall DataLoader and remove both binary and customer data including

configuration, enter the command

```
[dladmin@hdp2-w17 ~]./install_dl.sh -u -c
```

Configuring DataLoader

Modify the file `dataloader-env.sh` to reflect your Pivotal HD and Zookeeper cluster configuration. Either host names or their IP addresses can be used. This should be done as root (enter command proceeded by `sudo`.)

The file is located in `/usr/local/gphd/dataloader-2.0.4/conf`. For pseudo mode, edit the file in `/usr/local/gphd/dataloader-2.0.4/conf/pseudo/`

The following table gives the parameters that should be modified.dataloader

Table H.3 DataLoader Basic Parameters

Parameter	Function	Default value	Options
DATALOADER_ZK_QUORUM	List of Zookeeper nodes in comma separated format. This list must be the same as used in Zookeeper cluster configuration, (in the “zk.cfg” file).	Must be specified	N/A
DATALOADER_SERVICE_IP	IP address the services will bind to ("localhost" cannot be used).	Must be specified except for when running in standalone mode, and CLI is installed with service node	N/A
EXECUTOR_TYPE	Hadoop cluster version that DataLoader is running on. DataLoader supports Hadoop 1.0 (GPHD 1.2) and Pivotal HD 1.0.	mrv2	mrv1/mrv 2
HADOOP_INSTALL	The Hadoop installation directory. It is used in DataLoader to set right classpath, user must make sure this value is set correctly	${HADOOP_INSTALL} If user does not export this shell env variable or value is empty, it must be specified manually	N/A
HADOOP_CONF_DIR	The hadoop configuration directory.If you are using install_worker in tar ball to install worker, this value must be set. Otherwise optional	${HADOOP_CONF_DIR }If user does not export this shell env variable or value is empty, it must be specified manually	
DATALOADER_EXECUTOR_ CAPACITY	The total number of slots(MRv1) or containers (MRv2) that are available to DataLoader. This number, called “workers,” is set according to expected number of stream jobs, batch jobs, and capacity in the HDFS cluster. As an example, the number would be 20 on a 10 node Hadoop cluster to support up to 10 stream jobs, and leave capacity for copying 10s of TB of data in multiple simultaneous batch jobs. You must set this value for DataLoader to run correctly.	No default value, must be specified except for standalone mode.	

Table H.3 DataLoader Basic Parameters

Parameter	Function	Default value	Options
DATALOADER_DATA_DIR	The location to put DataLoader data files. This parameter is set during the pre-installation process using dl_cluster.conf.	For RPM installation, it is set in /etc/default/dataloader, value is: /var/lib/gphd/dataloader. In other case, it is ${DATALOADER_HOME}/data	
DATALOADER_LOG_DIR	The location to put the DataLoader log files. This parameter is set during the pre-installation process using dl_cluster.conf.	For RPM installation, it is set in /etc/default/dataloader, value is: /var/log/gphd/dataloader. In other case, it is set to: ${DATALOADER_HOME}/log	

Do not modify the following parameters.

- DATALOADER_HOME
- DATALOADER_CONF_DIR
- ZK_RUNTIME_DIR
- ENGINE_HOME

For pseudo-distributed mode, the following is a sample `dataloader-env.sh`:

```
export HADOOP_INSTALL=/usr/lib/gphd/hadoop
export HADOOP_CONF_DIR=/etc/gphd/hadoop/conf
export DATALOADER_ZK_QUORUM=localhost:12181
export DATALOADER_SERVICE_IP=localhost
export DATALOADER_EXECUTOR_CAPACITY=5
export EXECUTOR_TYPE=mrv1
```

Start and Stop DataLoader services

1. Change file ownership for the Linux account login; change the permission of the file /etc/shadow to be readable by the group root:

```
root# chmod g+r /etc/shadow
```

2. Login to the DataLoader master node as a user with sudo privileges or as dladmin, and go to the /bin directory at the install location:

```
$su - dladmin
$cd /usr/local/gphd/dataloader-2.0.4/bin/
```

Note: To reset the default `dlamin` user password, login as root and use "`passwd dladmin`" to change the password.

3. To start DataLoader service in standalone mode. The user DataLoader is created automatically during install.

```
sudo -u dataloader ./dataloader.sh start -s
```

To start service in distributed mode, enter:

```
sudo -u dataloader ./dataloader.sh start -d
```

To start service in pseudo-distributed mode, enter:

```
sudo -u dataloader ./dataloader.sh start -p
```

4. To stop DataLoader service:

```
sudo -u dataloader ./dataloader.sh stop all
```

Important: Verify that the Manager is running at Manager_IP:12380/manager.

Using the Linux service utility to start or stop DataLoader services

You can also use the Linux service utility to start or stop DataLoader services. However, this method is limited to starting DataLoader in distributed mode.

Note: Scheduler service should be started before the dataloader-manager service.

Start Services:

1. To start DataLoader services:

```
$ sudo service dataloader-scheduler start
$ sudo service dataloader-manager start
```

Note: The dataloader-scheduler service should be started before the dataloader-manager service.

Stop services:

To stop DataLoader services:

```
$ sudo service dataloader-manager stop
$ sudo service dataloader-scheduler stop
```

Note: In the above command, dataloader is a service user created during installation

Advanced Configurations

These configurations are used for performance tuning and are found in the `/usr/local/gphd/dataloader-2.0.4/conf/dataloader.xml` file.

Table H.4 Advanced Configuration Values

Parameter	Explanation	Default value	Options
dataloader.zk.address	Comma separated host/port list of zookeeper, this value should be changed, please change the DATALOADER_ZK_QUORUM in the *dataloader-env.sh* file.	${DATALOADER_ZK_QUORUM}	
dataloader.manager.service.port	The manager for the restful interface listening port;	12380	
dataloader.metrics.reporting.enable	This flag specifies whether to enable the metrics reporting mechanism for dataloader. Since progress monitoring depends on metrics reporting, disable this will also disable progress reporting.	true	true/false
dataloader.metrics.server.host	rpc host address for metrics reporting server. This parameter should be set to manager's host. This value should not be changed, if you want to change this value, please change DATALOADER_SERVICE_IP in the *dataloader-env.sh* file.	${DATALOADER_SERVICE_IP}	
dataloader.metrics.server.port	rpc port for metrics reporting server.	12322	
dataloader.scheduler.service.rest.port	The scheduler's restful interface listening port	12321	
dataloader.scheduler.service.rest.host *	The scheduler restful interface binding address. This value should not be changed, if you want to change this value, please change DATALOADER_SERVICE_IP in the *dataloader-env.sh* file.	${DATALOADER_SERVICE_IP}	

Table H.4 Advanced Configuration Values

Parameter	Explanation	Default value	Options
dataloader.scheduler.taskscheduler.host	The scheduler's host, worker will use this to contact scheduler for runtime task scheduling, This value should not be changed, if you want to change this value, please change DATALOADER_SERVICE_IP in the *dataloader-env.sh* file.	${DATALOADER_SERVICE_IP}	
dataloader.scheduler.taskscheduler.port	The scheduler's port. The worker will use this to contact the scheduler for runtime task scheduling,	12320	
dataloader.worker.reader.num	The number of reader threads will be started in each worker	3	
dataloader.worker.writer-pipeline.num	The number of writer/pipeline threads will be started in each worker	5	
dataloader.worker.buffer.num	The number of buffers which could be used in each worker	12	
dataloader.worker.buffer.size	The size of each buffer for worker	16 * 1024 * 1024 (16MB)	
dataloader.worker.streaming.server.memory.upper.limit	The maxiumum memory that the worker can use.	335544320	

Advanced features are used primarily for performance tuning.

Configure DataLoader CLI

To use the DataLoader CLI for managing jobs and datastores, it must be configured. The configuration file for CLI is located at:
`${dataloader_HOME}/conf/dataloader-cli.conf`.

It is a text file with key/value pair content and contains the following values:

Table H.5 CLI Configuration file values

Name	default values	Description
dataloader.api.url	http://localhost:12380/manager	This is the location of the DataLoader manager web address. If CLI is installed on the same machine as the DataLoader service package, this value doesn't need to be changed

Table H.5 CLI Configuration file values

Name	default values	Description
dataloader.zk.address	localhost:12181	This is the location of zookeeper servers, provided as a comma-separated list. This list should be the same as used in dataloader-env.sh earlier. Do not change if using standalone or pseudo-distributed modes, as the embedded-zookeeper address is the default.
dataloader.cli.queue.max.buffer;	32768	The maximum number of unacknowledged messages that the client can hold in memory. Will be overwritten at run time if the command line param --queue-size is specified.

If using pseudo-distributed mode, you can use default values, provided you are not using an external Zookeeper.

Glossary

B

bandwidth-throttling

The upper limit of aggregated network bandwidth consumed by a job.

batch job

A batch job is used for loading large numbers or large amounts of files/data in batch mode.

C

chunking

Splitting a file into chunks with fixed size, such as 640MB per chunk, and allocating data chunks to different data loading tasks.

compression

A file is compressed first before copying to the destination data store. The data is stored in the compressed format. DataLoader supports compression format: Snappy. Compression and chunking *cannot* be used at the same time.

connectlimited

Use this to restrict concurrent connections to a data store.

copy strategy

The copy strategy defines a method distributing and allocating the loading tasks to DataLoader workers. Different copy strategies are used, depending on the loading requirements. More than one strategy may apply to a loading scenario.

D

DataLoader job

A DataLoader job is defined by a JobSpec, which defines the data source and destination, readers, writers, and related entities. The job loading loading policy defines loading strategy (a.k.a Copy Strategy), and provides options for controlling how data will be loaded into the destination datastore.

DataLoader jobs support submit, start, suspend, resume, cancel, and query operations.

data store

An abstraction for a persistant store that can be used to store data, such as NFS, HDFS, HTTP, etc.

dynamic

Copy strategy. Load tasks are not pre-divided and allocated to worker machines Instead, the worker machine asks the scheduler for loading tasks at runtime when it has free resources.

G

glob

A specific pattern that provides a concise and flexible means to match a path to a datastore.

I

intelligent

The DataLoader scheduler decides the data copy strategy and policy based on the source data type and destination data store configuration.

J

job

A **Job** is submitted by DataLoaderMaster, and contains a number of task items, each of which can be processed individually.

Job Specification (JobSpec)

A JobSpec (also known as a Transfer Spec) is an XML-based file that defines a job being submitted.

L

locality

A worker is co-located with the data store. This describes an architecture where the source data store is an HDFS shared with the MapReduce JobTracker.

MVCC minimizes lock contention in order to allow for reasonable performance in multiuser environments by eschewing explicit locking methodologies used by traditional database systems. The main advantage to using the MVCC model of concurrency control rather than locking is that MVCC locks that are acquired for querying (reading) data do not conflict with locks acquired for writing data.

S

strategy

A method of distributing and allocating data loading tasks to DataLoader workers.

streaming job

A streaming job is used to load data from a data source where the data is generated dynamically. The typical sources of such data would be events, twitter feeds, streaming media (only HTTP-based streaming is currently supported), or RESTful/SOAP-based web service or other RPC endpoints. Streaming jobs are optimized for low latency, plus scalability for the number of concurrent feeds.

T

task item

A **task item** is the work unit which can be processed independently. Each **job partition** consists of one or more **task item**s.

Transfer Spec

See "Job Specification (JobSpec)".

Transfer Policy

A transfer policy is a list of properties that control how the loading job should be executed.

U

uniform

Uniformly distributes the copying blocks to the DataLoader workers, based on data size.