

Simplify the Hadoop® ETL design process

The Dell™ | Cloudera® | Syncsort® solution for Hadoop enabled less-experienced users to develop and deploy Hadoop ETL jobs in less than a week.*



47.8%

less time

to design an ETL job with slowly changing dimensions (SCD)

55.5%

less time

to design an ETL job for performing data validation

58.8%

less time

to design an ETL job that integrates mainframe datasets

*compared to a senior engineer using DIY open-source tools

Four Dell PowerEdge™ R730xd servers and two Dell PowerEdge R730 servers, powered by the Intel® Xeon® processor E5-2600 v3 product family, in a Hadoop cluster.



cloudera®



syncsort

Many companies are adopting Hadoop solutions to handle large amounts of data stored across clusters of servers. Hadoop is a distributed, scalable approach to managing Big Data that is very powerful and can bring great value to organizations. Companies use extract, transform, and load (ETL) jobs to bring together data from many different applications or systems on different hardware in order to modify or adjust the data in some way, and then put it into a new format that they can mine for useful information.

Using traditional ETL can require highly experienced, expensive, and hard-to-find programmers to create jobs in order to extract data. Dell, Cloudera, and Syncsort offer an integrated Hadoop ETL solution that allows entry-level technicians—after only a few days of training—to perform the same tasks that these Hadoop specialists perform, often even more quickly. In our tests, we found that with the unique design of the Dell | Cloudera | Syncsort solution can allow an end user with little experience using Hadoop to develop and deploy optimized ETL jobs up to 58.8 percent faster than an expert-driven do-it-yourself (DIY) solution deployed using open-source tools.



SAVE TIME CREATING ETL JOBS WITH THE DELL | CLUDERA | SYNCSORT SOLUTION

Extract, Transform, and Load

ETL refers to the following process in database usage and data warehousing:

- Extract the data from multiple sources
- Transform the data so it can be stored properly for querying and analysis
- Load the data into the final database, operational data store, data mart, or data warehouse

Hadoop implementations suffer from several barriers to effectiveness. They often have primitive integration with infrastructure, and today we find there is currently a lack of available talent to run Hadoop clusters and perform data ingest and processing tasks using the cluster.¹ The Dell | Syncsort solution offers help with both of these problems.

The Dell | Cloudera | Syncsort solution is a reference architecture that offers a reliable, tested configuration that incorporates Dell hardware on the Cloudera Hadoop platform, with Syncsort's DMX-h ETL software. The Dell | Cloudera | Syncsort reference architecture includes four Dell PowerEdge R730xd servers and two Dell PowerEdge R730 servers, powered by the Intel Xeon processor E5-2600 v3 product family.

For organizations that want to optimize their data warehouse environments, the Dell | Cloudera | Syncsort reference architecture can greatly reduce the time needed to deploy Hadoop when using the included setup and configuration documentation as well as the validated best practices. Leveraging the Syncsort DMX-h software means Hadoop ETL jobs can be developed using a graphical interface in a matter of hours, with minor amounts of training, and with no need to spend days developing code. The Dell | Cloudera | Syncsort solution also offers professional services with Hadoop and ETL experts to help fast track your project to successful completion.²

To understand how fast and easy designing ETL jobs with the Dell | Cloudera | Syncsort solution can be, we had an entry-level technician and a highly experienced Hadoop expert work to create three Hadoop ETL jobs using different approaches to meet the goals of several use cases.

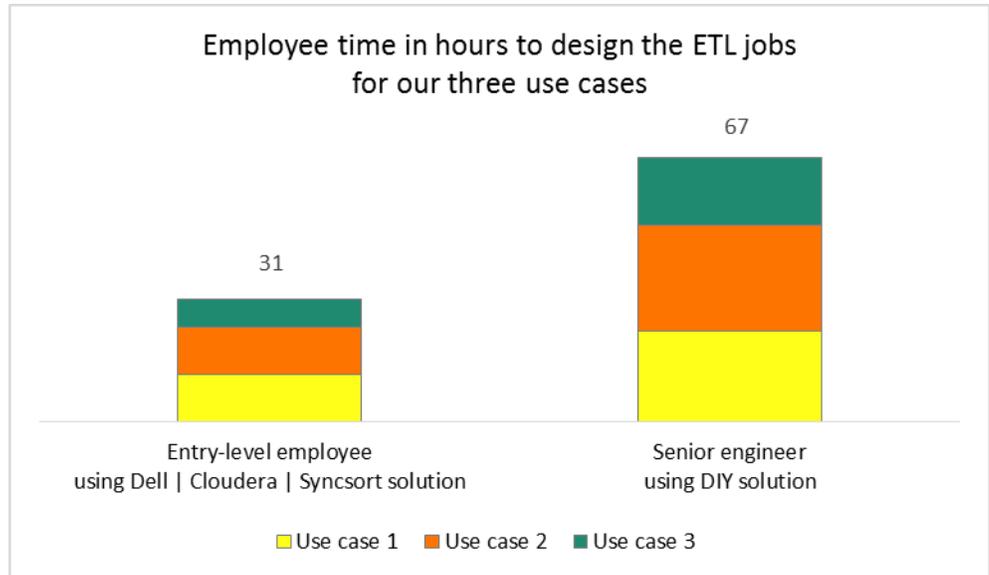
The entry-level worker, who had no familiarity with Hadoop and less than one year of general server experience, used Syncsort DMX-h to carry out these tasks. Our expert had 18 years of experience designing, deploying, administering, and benchmarking enterprise-level relational database management systems (RDBMS). He has deployed, managed, and benchmarked Hadoop clusters, covering several Hadoop distributions and several Big Data strategies. He designed and created the use cases using only free open-source DIY tools.

¹ Source: survey of attendees for the 2014 Gartner webinar Hadoop 2.0 Signals Time for Serious Big Data Consideration. www.informationweek.com/big-data/software-platforms/cloudera-trash-talks-with-enterprise-data-hub-release/d/d-id/1113677

² Learn more at en.community.dell.com/dell-blogs/dell4enterprise/b/dell4enterprise/archive/2015/06/09/fast-track-data-strategies-etl-offload-hadoop-reference-architecture

The implementation experience of these two workers showed us that using the Dell | Cloudera | Syncsort solution was faster, easier, and—because a less experienced employee could use it to create ETL jobs—far less expensive to implement. In this cost analysis, we apply those findings to a hypothetical large enterprise.

Figure 1: Our entry-level employee completed the three ETL design jobs using the Dell | Cloudera | Syncsort solution in less than half the time the much more experienced engineer required using other tools. (Lower numbers are better.)



As Figure 1 shows, the beginner using the Dell | Cloudera | Syncsort solution was able to complete the three ETL design jobs in a total of 31 hours, whereas the senior engineer using open-source tools needed 67 hours—more than twice as long. Note that in addition to actually coding/designing the use cases, both of the workers spent time familiarizing themselves with the requirements, revising to correct output issues, and validating results.

Along with the savings³ that come from having a less highly compensated employee perform the design work more quickly, the Dell | Cloudera | Syncsort solution offers another avenue to cost-effectiveness: performance. Additional testing in the PT labs⁴ revealed that due to their extreme efficiency, the ETL jobs our entry-level worker created using Syncsort DMX-h ran more quickly than those our highly compensated expert created. This can lead to savings in server utilization.

In this paper, we provide a quick overview of the Dell | Cloudera | Syncsort solution and then discuss the experience the entry-level technician had while receiving training and using the Dell | Cloudera | Syncsort solution.

³ Cost advantages of Hadoop ETL offload with the Intel processor-powered Dell | Cloudera | Syncsort solution http://www.principledtechnologies.com/Dell/Dell_Cloudera_Syncsort_cost_0715.pdf

⁴ Performance advantages of Hadoop ETL offload with the Intel processor-powered Dell | Cloudera | Syncsort solution www.principledtechnologies.com/Dell/Dell_Cloudera_Syncsort_performance_0715.pdf

ABOUT SYNCSORT DMX-h

Syncsort DMX-h is a high-performance data integration software that runs natively in Hadoop, providing everything needed to collect, prepare, blend, transform, and distribute data. DMX-h, with its Intelligent Execution, allows users to graphically design sophisticated data flows once and deploy on any compute framework (Apache MapReduce, Spark, etc. on premise or in the cloud), future-proofing the applications while eliminating the need for coding.

Using an architecture that runs ETL processing natively in Hadoop, without code generation, Syncsort DMX-h lets users maximize performance without compromising on the capabilities and typical use cases of conventional ETL tools. In addition, the software packages' industrial-grade capabilities to deploy, manage, monitor, and secure your Hadoop environment.

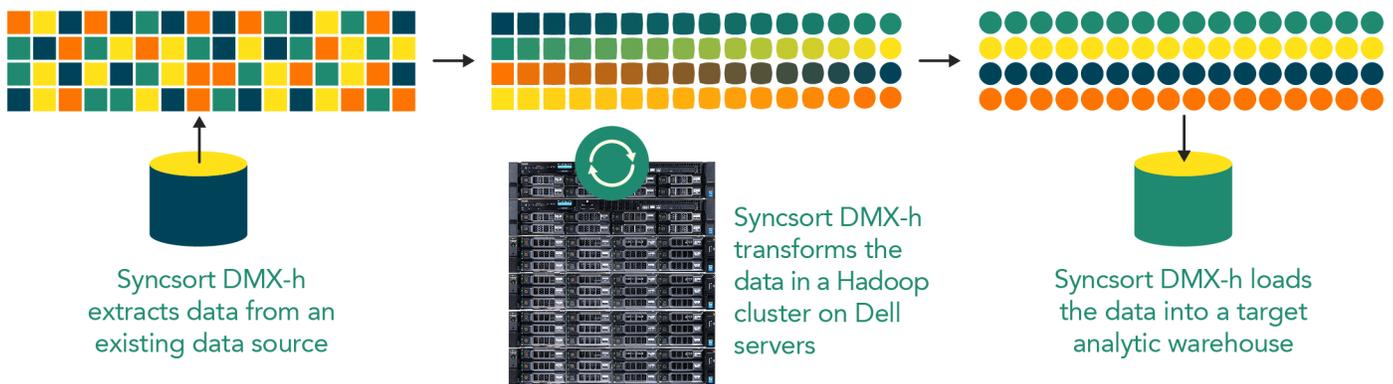


Figure 2: How the Dell | Cloudera | Syncsort solution for Hadoop works.

Syncsort SILQ®

Syncsort SILQ is a technology that pairs well with DMX-h. SILQ is a SQL offload utility designed to help users visualize and move their expensive data warehouse (SQL) data integration workloads into Hadoop. SILQ supports a wide range of SQL flavors and can parse thousands of lines of SQL code in seconds, outputting logic flowcharts, job analysis, and DMX-h jobs. SILQ has the potential to take an overwhelming SQL workload migration process and make it simple and efficient.

About Dell | Cloudera | Syncsort training and professional services

An integral part of the Dell | Cloudera | Syncsort solution is the on-site training that the company makes available for a fee. This training typically lasts four days. However, because of our tight time frame, our technician worked with the trainer for two and a half intensive days. The points below are based on the reflections of the technician:

- The training sessions were a mix of lecture, lessons, and question and answer.
- The technician logged into Syncsort's training labs where VMs were set up with Hadoop clusters ready for practicing on.
- The technician was able to ask the instructor anything, which helped him get a sense of best practices and how other people are currently using the product.
- In addition to learning how to use DMX-h, the technician also learned how to use Syncsort's documentation effectively, which made building jobs much easier.
- By the end of the training, the technician felt he had been well exposed to every area of Syncsort DMX-h.

Another important component of the Dell | Cloudera | Syncsort solution is ongoing enablement assistance that the company makes available for a fee. (Technical support to answer questions about the product is included with the purchase price.) The points below are based on the reflections of the technician:

- After setting up his first job in a matter of hours, our technician sought approval from Syncsort. Their enablement assistance team helped him finalize and tweak his job for performance.
- When our technician had a question about his output being formatted correctly, the Syncsort team looked at his job definitions and provided feedback.

THE ETL JOBS OUR WORKERS DESIGNED

In our tests, both the entry-level technician and the senior engineer used the same source dataset on the HDFS cluster, generated from the TPC-H dataset at scale factor 200. The entry-level technician utilized Syncsort DMX-h to develop the ETL job for each use case, while the senior engineer had the flexibility to utilize the publically available open-source tools of their choice. The senior engineer chose to utilize Apache® Pig because of its procedural nature rather than other commonly used tools, such as Apache Hive.

For each of the three use cases, we compared the resulting output of the two ETL jobs to ensure that they achieved the same business goals. See [Appendix A](#) for more details on our ETL jobs.

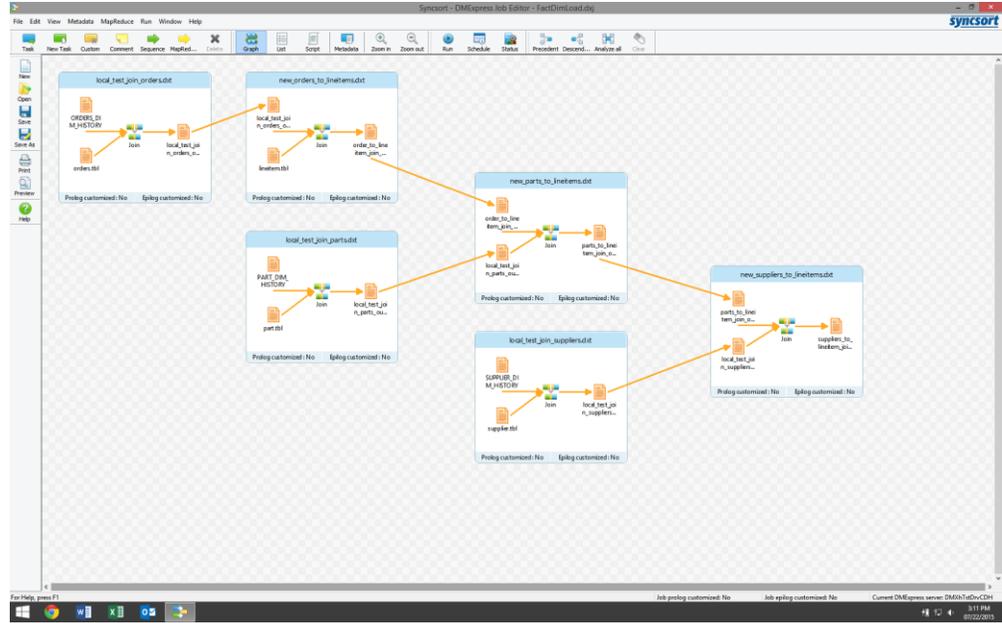
Use case 1: Fact dimension load with Type 2 Slowly Changing Dimensions (SCD)

Businesses often feed their Business Intelligence decisions using the Dimensional Fact Model, which uses a set of dimensional tables (tables with active and historical data about a specific category, such as Geography or Time) to feed a fact table (a table that joins the active records of all the dimensional tables on a common field) summarizing current business activity. While some dimension tables never change, some often experience updates. The dimension tables must be up-to-date to feed the fact table correctly, and many businesses would like to keep historical data as well for archival purposes.

A common method to identify changes in data is to compare historical and current data using Changed Data Capture (CDC). A common method for updating records while retaining the outdated record is Type 2 Slowly Changing Dimensions (SCD). We used outer joins and conditional formatting to implement CDC and identify changed records. We then implemented Type 2 SCD to mark outdated records with an end-date and insert the updated data as a new, current record (in our case we used a perpetual end-date of 12/31/9999 to specify a current record). We then fed the information from the dimensional tables downstream to a fact table that could be queried for Business Intelligence.

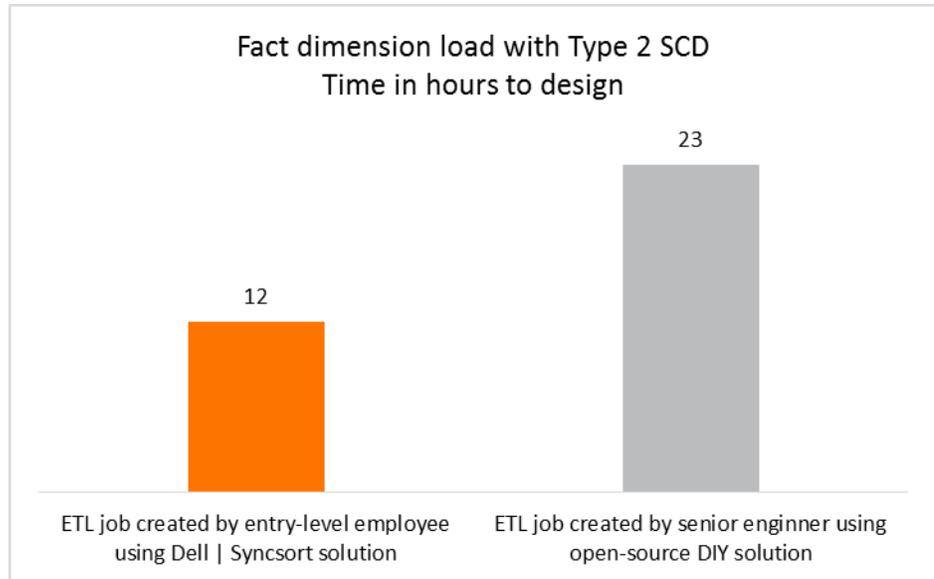
Figure 3 shows the Syncsort DMX-h job editor with the completed use case 1 design. The individual boxes represent tasks, while the arrows between the tasks represent data flow.

Figure 3: Syncsort DMX-h job editor showing the completed use case 1 design.



Both the entry-level technician using the Dell | Cloudera | Syncsort solution and the senior engineer using open-source DIY tools were able to design effective jobs to perform Type 2 Slowly Changing Dimensions to feed the Dimensional Fact Model. The beginner was able to design the job 47.8 percent faster than the senior engineer (see Figure 4).

Figure 4: Time to design the Fact dimension load with Type 2 SCD use case. (Lower numbers are better.)



Use case 2: Data validation and pre-processing

Data validation is an important part of any ETL process. It is important for a business to ensure that only usable data makes it into their data warehouse. And, that when data is unusable, it's important to mark that unusable data with an error message explaining why it's invalid. Data validation can be performed with conditions and filters to check for and discard data that is badly formatted or nonsensical.

Figure 5 shows the Syncsort DMX-h job editor with the completed use case 2 design, which pictures the metadata panel showing a sample of the conditions used to reformat and validate the data.

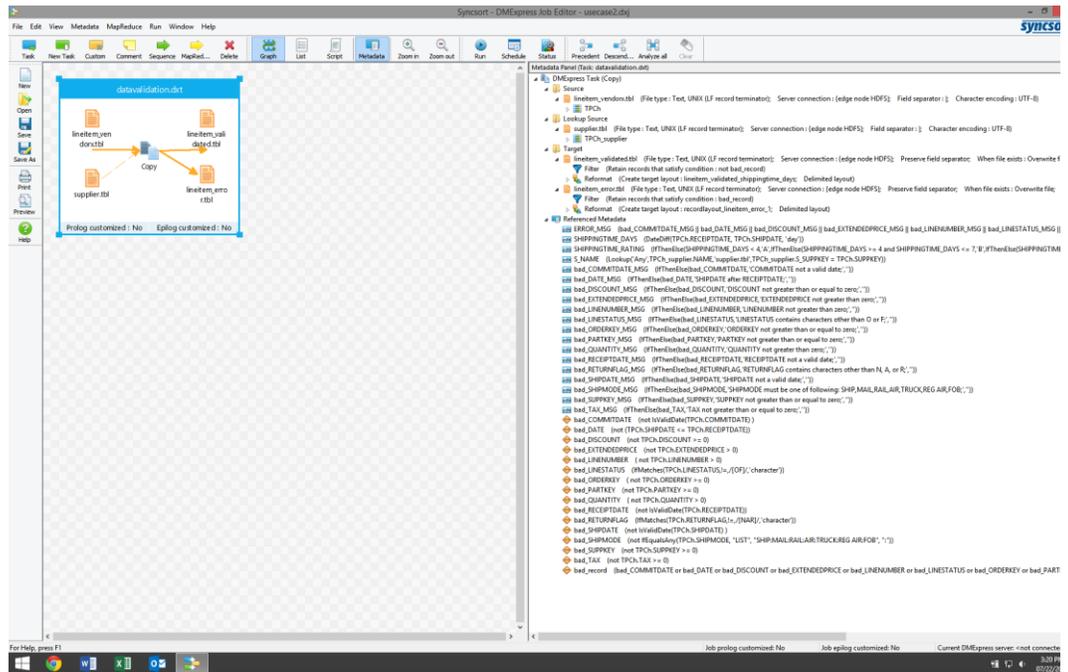
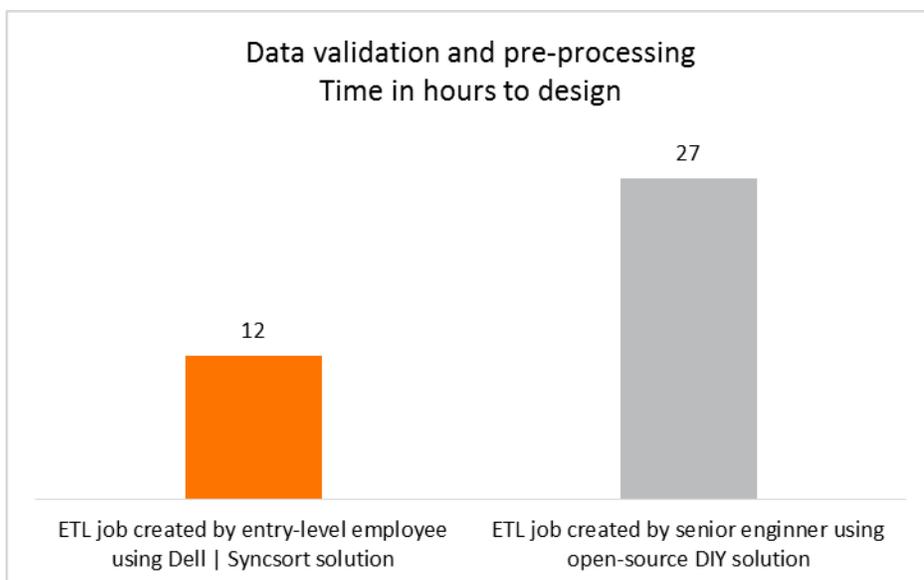


Figure 5: Syncsort DMX-h job editor showing the completed use case 2 design.

Both the entry-level technician using the Dell | Cloudera | Syncsort solution and the senior engineer using open-source DIY tools were able to design effective jobs to perform accurate data validation, but the beginner was able to design the job 55.6 percent faster than the senior engineer using Apache Pig (see Figure 6). The Syncsort DMX-h graphical user interface made it easy to quickly try new filters and sample the outputs. Being able to rapidly prototype the data validation job meant less time spent revising.

Figure 6: Time to create the Data validation and pre-processing use case. (Lower numbers are better.)



Use case 3: Vendor mainframe file integration

The need to reformat heterogeneous data sources into a more useable format is a common requirement of businesses with many varied data sources. Often companies have high-value data locked away in the mainframe and want to assimilate that data into the cluster. Some mainframe data types use a COBOL copybook to specify field formats. Syncsort DMX-h makes the process of reformatting mainframe data sources easy by allowing the user to link a COBOL copybook into metadata and automatically interpret the flat file’s record layout—one of many data integration features Syncsort DMX-h offers. Syncsort DMX-h enabled an inexperienced IT person with no prior exposure to mainframe-formatted files to interpret these legacy data types, all without disturbing the integrity of the original data.

Figure 7 shows the Syncsort DMX-h job editor with the completed use case 3 design. The task shows a single mainframe-formatted data source being split into two ASCII-formatted outputs.

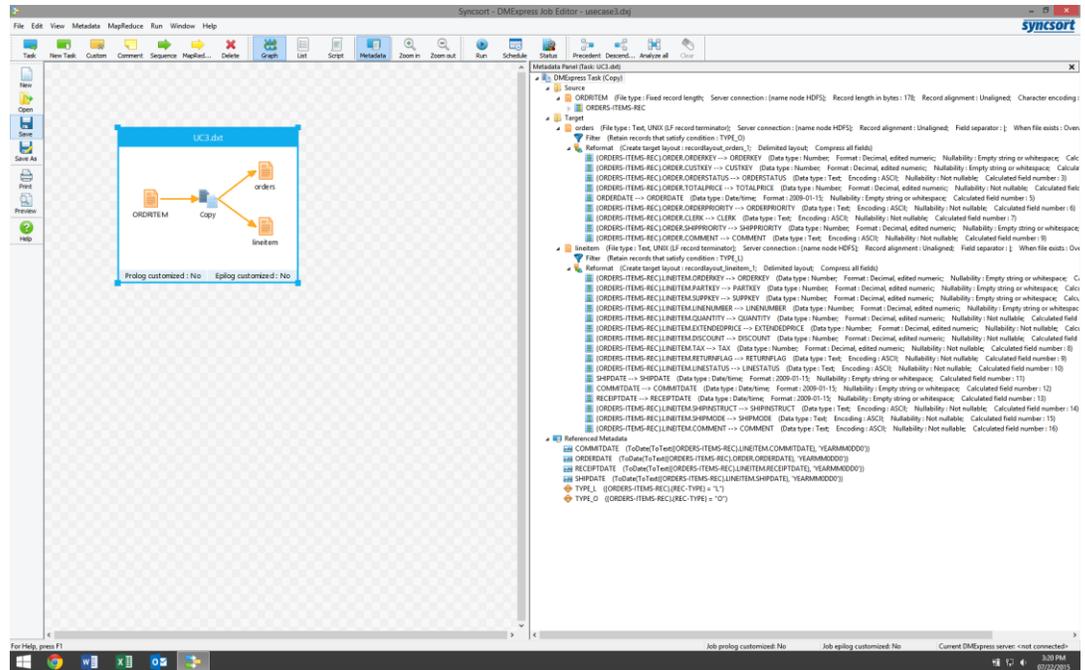


Figure 7: Syncsort DMX-h job editor showing the completed use case 3 design.

Both the entry-level technician using the Dell | Cloudera | Syncsort solution and the senior engineer using open source DIY tools were able to design effective jobs to reformat the mainframe data successfully, but the beginner was able to design the job 58.8 percent faster (see Figure 8).

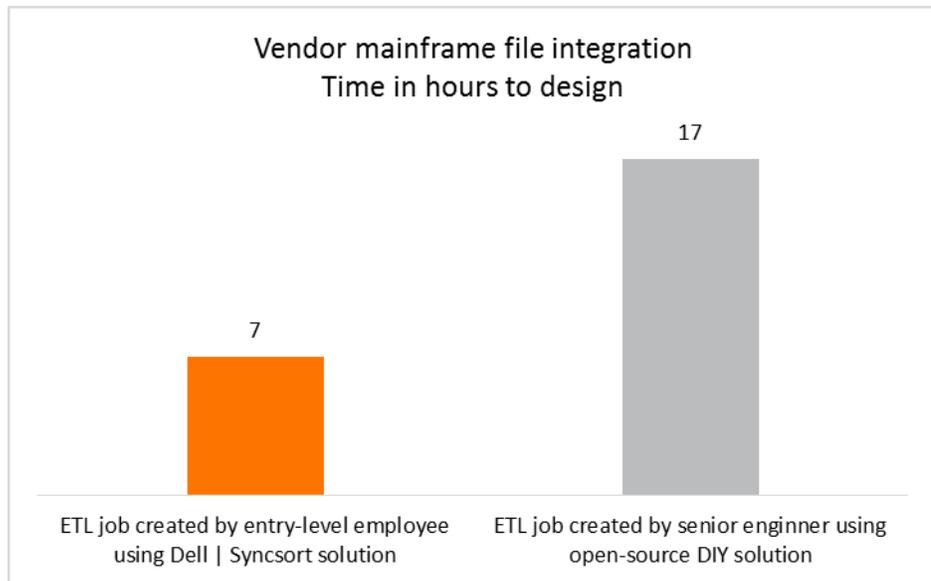


Figure 8: Time to create the Vendor mainframe file integration use case. (Lower numbers are better.)

CONCLUSION

High-level Hadoop analysis requires custom solutions to deliver the data that you need and the amount of time that even senior engineers require to create ETL jobs in a DIY hardware and software situation, can be substantial.

We found that the Dell | Cloudera | Syncsort solution was so easy to use that an entry-level employee could use it to create optimized ETL jobs after only a few days of training. And he could do it quickly—our technician, who had no previous experience using Hadoop, developed three optimized ETL jobs in 31 hours. That is less than half of the 68 hours our expert with years of Hadoop experience needed to create the same jobs using open source tools.

Using the Dell | Cloudera | Syncsort solution means that your organization can implement a Hadoop solution using employees already on your staff rather than trying to recruit expensive, difficult-to-find specialists. Not only that, but the projects can be completed in a fraction of the time. This makes the Dell | Cloudera | Syncsort solution a winning business proposition.

APPENDIX A – HOW WE TESTED

Installing the Dell | Cloudera Apache Hadoop Solution

We installed Cloudera Hadoop (CDH) version 5.4 onto our cluster by following the “Dell | Cloudera Apache Hadoop Solution Deployment Guide – Version 5.4” with some modifications. The following is a high-level summary of this process.

Configuring the Dell Force10 S55 and Dell PowerConnect S4810 switches

We used the Dell Force10 S55 switch for 1GbE external management access from our lab to the Edge Node. We configured two Dell PowerConnect S4810 switches for redundant 10GbE cluster traffic.

Configuring the BIOS, firmware, and RAID settings on the hosts

We used the Dell Deployment Tool Kit to configure our hosts before OS installation. We performed these steps on each host.

1. Boot into the Dell DTK USB drive using BIOS boot mode.
2. Once the CentOS environment loads, choose the node type (infrastructure or storage), and enter the iDRAC connection details.
3. Allow the system to boot into Lifecycle Controller and apply the changes. Once this is complete, the system will automatically reboot once more.

Installing the OS on the hosts

We installed CentOS 6.5 using a kickstart file with the settings recommended by the Deployment Guide. We performed these steps on each node.

1. Boot into a minimal CentOS ISO and press Tab at the splash screen to enter boot options.
2. Enter the kickstart string and required options, and press Enter to install the OS.
3. When the OS is installed, run yum updates on each node, and reboot to fully update the OS.

Installing Cloudera Manager and distributing CDH to all nodes

We used Installation Path A in the Cloudera support documentation to guide our Hadoop installation. We chose to place Cloudera Manager on the Edge Node so that we could easily access it from our lab network.

1. On the Edge Node, use `wget` to download the latest `cloudera-manager-installer.bin`, located on `archive.cloudera.com`.
4. Run the installer and select all defaults.
5. Navigate to Cloudera Manager by pointing a web browser to `http://<Edge_Node_IP_address>:7180`.
6. Log into Cloudera Manager using the default credentials `admin/admin`.
7. Install the Cloudera Enterprise Data Hub Edition Trial with the following options:
 - a. Enter each host’s IP address.
 - b. Leave the default repository options.
 - c. Install the Oracle Java SE Development Kit (JDK).
 - d. Do not check the single user mode checkbox.
 - e. Enter the root password for host connectivity.
8. After the Host Inspector checks the cluster for correctness, choose the following Custom Services:
 - a. HDFS
 - b. Hive

- c. Hue
- d. YARN (MR2 Included)

9. Assign roles to the hosts using Figure 9.

Service	Role	Node(s)
HBase		
	Master	nn01
	HBase REST Server	nn01
	HBase Thrift Server	nn01
	Region Server	nn01
HDFS		
	NameNode	nn01
	Secondary NameNode	en01
	Balancer	en01
	HttpFS	nn01
	NFS Gateway	nn01
	DataNode	dn[01-04]
Hive		
	Gateway	[all nodes]
	Hive Metastore Server	en01
	WebHCat Server	en01
	HiveServer2	en01
Hue		
	Hue Server	en01
Impala		
	Catalog Server	nn01
	Impala StateStore	nn01
	Impala Daemon	nn01
Key-value Store Indexer		
	Lily Hbase Indexer	nn01
Cloudera Management Service		
	Service Monitor	en01
	Activity Monitor	en01
	Host Monitor	en01
	Reports Manager	en01
	Event Server	en01
	Alert Publisher	en01
Oozie		
	Oozie Server	nn01
Solr		
	Solr Server	nn01
Spark		
	History Server	nn01
	Gateway	nn01

Service	Role	Node(s)
Sqoop 2		
	Sqoop 2 Server	nn01
YARN (MR2 Included)		
	ResourceManager	nn01
	JobHistory Server	nn01
	NodeManager	dn[01-04]
Zookeeper		
	Server	nn01, en01, en01

Figure 9: Role assignments.

- At the Database Setup screen, copy down the embedded database credentials and test the connection. If the connections are successful, proceed through the wizard to complete the Cloudera installation.

Installing the Syncsort DMX-h environment

Installation of the Syncsort DMX-h environment involves installing the Job Editor onto a Windows server, distributing the DMX-h parcel to all Hadoop nodes, and installing the `dmxd` service onto the NameNode. We used the 30-day trial license in our setup.

Installing Syncsort DMX-h onto Windows

We used a Windows VM with access to the NameNode to run the Syncsort DMX-h job editor.

- Run `dmexpress_8-1-0_windows_x86.exe` on the Windows VM and follow the wizard steps to install the job editor.

Distributing the DMX-h parcel via Cloudera Manager

We downloaded the DMX-h parcel to the Cloudera parcel repository and used Cloudera Manager to pick it up and send it to every node.

- Copy `dmexpress-8.1.7-el6.parcel_en.bin` to the EdgeNode and set execute permissions for the `root` user.
- Run `dmexpress-8.1.7-el6.parcel_en.bin` and set the extraction directory to `/opt/cloudera/parcel-repo`.
- In Cloudera Manager, navigate to the Parcels section and distribute the DMExpress parcel to all nodes.

Installing the dmxd daemon on the NameNode

We placed the `dmxd` daemon on the NameNode in order to have it in the same location as the YARN ResourceManager.

- Copy `dmexpress-8.1.7-1.x86_64_en.bin` to the NameNode and set execute permissions for the `root` user.
- Run `dmexpress-8.1.7-el6.parcel_en.bin` to install the `dmxd` daemon.

Post-install configuration

We made a number of changes to the cluster in order to suit our environment and increase performance.

Relaxing HDFS permissions

We allowed the `root` user to read and write to HDFS, in order to simplify the process of performance testing.

- In Cloudera Manager, search for “Check HDFS Permissions” and uncheck the HDFS (Service-Wide) checkbox.

Setting YARN parameters

We made a number of parameter adjustments to increase resource limits for our map-reduce jobs. These parameters can be found using the Cloudera Manager search bar. Figure 10 shows the parameters we changed.

Parameter	New value
yarn.nodemanager.resource.memory-mb	80 GiB
yarn.nodemanager.resource.cpu-vcores	35
yarn.scheduler.maimum-allocation-mb	16 GiB

Figure 10: YARN resource parameter adjustments.

Custom XML file for DMX-h jobs

We created an XML file to set cluster parameters for each job. In the Job Editor, set the environment variable DMX_HADOOP_CONF_FILE to the XML file path. The contents of the XML file are below.

```
<?xml version="1.0"?>

<configuration>

<!-- Specify map vcores resources -->
<property>
<name>mapreduce.map.cpu.vcores</name>
<value>2</value>
</property>

<!-- Specify reduce vcores resources -->
<property>
<name>mapreduce.reduce.cpu.vcores</name>
<value>4</value>
</property>

<!-- Specify map JVM Memory resources -->
<property>
<name>mapreduce.map.java.opts</name>
<value>-Xmx2048m</value>
</property>

<!-- Specify reduce JVM Memory resources -->
<property>
<name>mapreduce.reduce.java.opts</name>
<value>-Xmx7168m</value>
</property>

<!-- Specify map Container Memory resources -->
<property>
```

```

<name>mapreduce.map.memory.mb</name>
<value>2560</value>
</property>

<!-- Specify reduce Container Memory resources -->
<property>
<name>mapreduce.reduce.memory.mb</name>
<value>8704</value>
</property>

<!-- Specify reducers to be used -->
<property>
<name>mapreduce.job.reducers</name>
<value>32</value>
</property>

</configuration>

```

Creating the Syncsort DMX-h use cases

In our testing, we measured the time required to design and create DMX-h jobs for three use cases. Screenshots of the DMX-h jobs for each use case appear below.

Use case 1: Fact dimension load with Type 2 Slowly Changing Dimensions (SCD)

We used outer joins and conditional reformatting to implement Type 2 SCD for Use Case 1. Figure 11 shows the UC1 job layout.

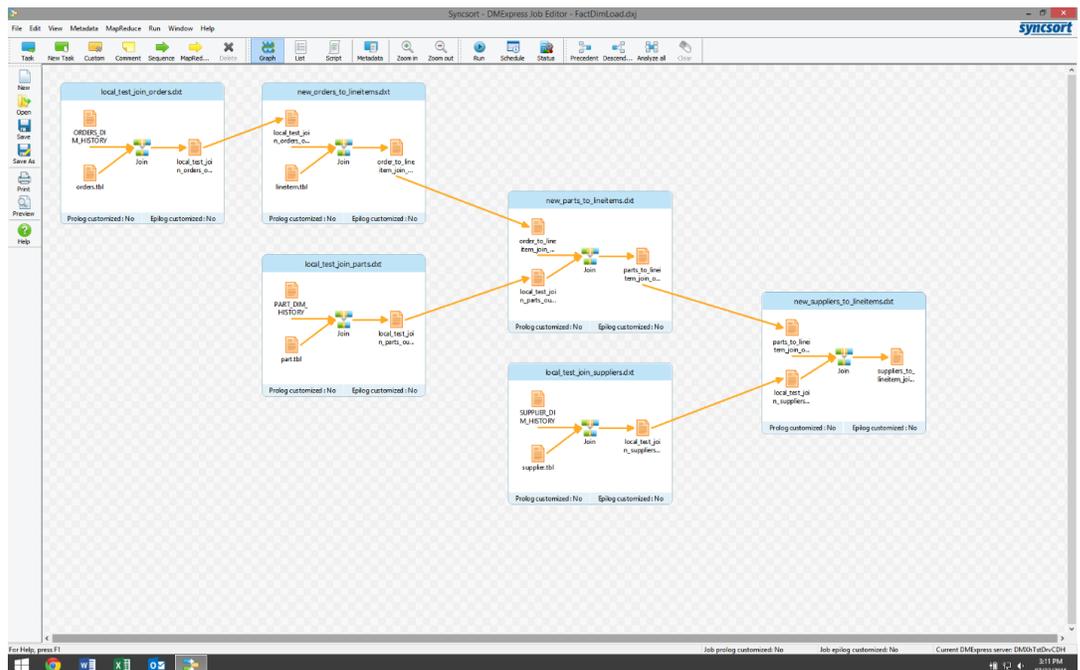


Figure 11: Use Case 1 job layout.

Creating the DIY use cases

We used Pig, Python and Java to implement the DIY approaches to each use case. The DIY code for each use case appears below.

Use case 1: Fact dimension load with Type 2 Slowly Changing Dimensions (SCD)

```
set pig.maxCombinedSplitSize 2147483648
set pig.exec.mapPartAgg true

-- uc1

-----
-----
-- parameters and constants

-- input and output files
%DECLARE IN_O_UPDATES '/UC1/200/Source/orders.tbl'
%DECLARE IN_P_UPDATES '/UC1/200/Source/part.tbl'
%DECLARE IN_S_UPDATES '/UC1/200/Source/supplier.tbl'
%DECLARE IN_O_HISTORY '/UC1/200/Source/ORDERS_DIM_HISTORY'
%DECLARE IN_P_HISTORY '/UC1/200/Source/PART_DIM_HISTORY'
%DECLARE IN_S_HISTORY '/UC1/200/Source/SUPPLIER_DIM_HISTORY'
%DECLARE IN_LINEITEM '/UC1/200/Source/lineitem.tbl'
%DECLARE OUT_O_HISTORY '/DIY/200/UC1/ORDERS_DIM_HISTORY'
%DECLARE OUT_P_HISTORY '/DIY/200/UC1/PART_DIM_HISTORY'
%DECLARE OUT_S_HISTORY '/DIY/200/UC1/SUPPLIER_DIM_HISTORY'
%DECLARE OUT_FACT '/DIY/200/UC1/SOLD_PARTS_DIM_FACT'

-- "interior" fields for the i/o tables. needed to assist with column projections
%DECLARE HI_O_FIELDS
'ho_custkey,ho_orderstatus,ho_totalprice,ho_orderdate,ho_orderpriority,ho_clerk,ho_shippriority'
%DECLARE HI_P_FIELDS
'hp_name,hp_mfgr,hp_brand,hp_type,hp_size,hp_container,hp_retailprice'
%DECLARE HI_S_FIELDS 'hs_name,hs_address,hs_nationkey,hs_phone,hs_acctbal'
%DECLARE UP_O_FIELDS
'uo_custkey,uo_orderstatus,uo_totalprice,uo_orderdate,uo_orderpriority,uo_clerk,uo_shippriority'
%DECLARE UP_P_FIELDS
'up_name,up_mfgr,up_brand,up_type,up_size,up_container,up_retailprice'
%DECLARE UP_S_FIELDS 'us_name,us_address,us_nationkey,us_phone,us_acctbal'

-- Option to use replicated JOIN for the supplier name lookup.
-- use it
%DECLARE USE_REP_JOIN 'USING \'REPLICATED\'' ;
-- don't use it
-- %DECLARE USE_REP_JOIN ' ' ;

-- tag in end-date fieldstop signify an active records
%DECLARE OPEN_REC '12/31/9999'

-- tags for start/end date fields
%DECLARE TODAY_REC `date +%m/%d/%Y`
%DECLARE YESTE_REC `date --date="1 days ago" +%m/%d/%Y`

IMPORT 'uc1_macros.pig';
```

```

-----
-----

-- supplier, new history

supplier = update_history('$IN_S_UPDATES', '$IN_S_HISTORY', '$SUP_S_FIELDS',
'$HI_S_FIELDS');
STORE supplier INTO '$OUT_S_HISTORY' USING PigStorage('|');

-- part, new history

part = update_history('$IN_P_UPDATES', '$IN_P_HISTORY', '$SUP_P_FIELDS', '$HI_P_FIELDS');
STORE part INTO '$OUT_P_HISTORY' USING PigStorage('|');

-- orders, new history

orders = update_history('$IN_O_UPDATES', '$IN_O_HISTORY', '$SUP_O_FIELDS',
'$HI_O_FIELDS');
STORE orders INTO '$OUT_O_HISTORY' USING PigStorage('|');

-----
-----

-- drop expired records

supplier = FILTER supplier BY h_enddate == '$OPEN_REC';
part      = FILTER part      BY h_enddate == '$OPEN_REC';
orders    = FILTER orders    BY h_enddate == '$OPEN_REC';

-- data for fact table

lineitem = LOAD '$IN_LINEITEM' USING PigStorage('|') AS (
    l_orderkey,l_partkey,l_suppkey,
    l_linenumbr,l_quantity,l_extendedprice,l_discount,l_tax,l_returnflag,
    l_linestatus,l_shipdate,l_commitdate,
    l_receiptdate,l_shipinstruct,l_shipmode,l_comment
);

-- dereference supplier and save required keys (drop l_suppkey)

lineitem = JOIN lineitem by l_suppkey LEFT OUTER, supplier by h_key;
lineitem = FOREACH lineitem GENERATE
    l_orderkey,l_partkey,
    hs_name,hs_nationkey,hs_acctbal,supplier::h_startdate,l_linenumbr .. l_comment;

-- dereference part and save required keys (drop l_partkey)

lineitem = JOIN lineitem by l_partkey LEFT OUTER, part by h_key;
lineitem = FOREACH lineitem GENERATE
    l_orderkey,
    hp_name,hp_mfgr,hp_brand,hp_type,hp_retailprice,part::h_startdate,hs_name ..
    l_comment;

-- dereference orders and save required keys (drop l_orderkey)

```

```

lineitem = JOIN lineitem by l_orderkey LEFT OUTER, orders by h_key;
lineitem = FOREACH lineitem GENERATE
    ho_custkey,ho_orderstatus,ho_totalprice,ho_orderdate,orders::h_startdate,hp_name ..
l_comment;

```

```

-----
-----

```

```

STORE lineitem INTO '$OUT_FACT' USING PigStorage('|');

```

```

ucl_macros.pig

```

```

DEFINE update_history(in_updates, in_history, update_fields, history_fields) RETURNS
history {
    -- update tables
    updates = LOAD '$in_updates' USING PigStorage('|') AS (
        u_key, $update_fields, u_comment
    );
    -- historical tables
    historical = LOAD '$in_history' USING PigStorage('|') AS (
        h_key, $history_fields, h_comment, h_startdate:chararray,h_enddate:chararray
    );
    -- remove expired records from the historical data and save for final table
    SPLIT historical INTO
        in_unexpired IF (h_enddate matches '^$OPEN_REC$'),
        in_expired OTHERWISE;
    -- full join by primary key to determine matches and left/right uniques
    joined = JOIN
        updates by u_key FULL OUTER,
        in_unexpired by h_key;
    SPLIT joined INTO
        new IF h_key IS NULL, -- totally new entry from updates
        old IF u_key IS NULL, -- unmatched historical entry
        matched OTHERWISE; -- match primary key: either redundant or updated entry
    -- format new and old entries for output
    new = FOREACH new GENERATE u_key .. u_comment, '$TODAY_REC', '$OPEN_REC';
    old = FOREACH old GENERATE h_key .. h_enddate;
    -- find updated entries
    SPLIT matched INTO
        updates IF (TOTUPLE($update_fields, u_comment) != TOTUPLE($history_fields,
h_comment) ),
        redundant OTHERWISE;
    -- format redundant entries for output
    redundant = FOREACH redundant GENERATE h_key .. h_enddate;
    -- updated entry: expire historical; tag update
    modified = FOREACH updates GENERATE u_key .. u_comment, '$TODAY_REC', '$OPEN_REC';
    expired = FOREACH updates GENERATE h_key .. h_startdate, '$YESTE_REC';
    -- combine the record classes
    $history = UNION in_expired, new, old, redundant, modified, expired;
};

```

Use case 2: Data validation and pre-processing

```
set pig.maxCombinedSplitSize 1610612736
set pig.exec.mapPartAgg true

-----
-----
-- parameter and constants definitions

-- input and output files
%DECLARE IN_LINEITEM_VENDOR '/UC2/200/Source/lineitem_vendorx.tbl'
%DECLARE IN_SUPPLIER        '/UC2/200/Source/supplier.tbl'
%DECLARE OUT_GOOD           '/DIY/200/UC2/lineitem_validated'
%DECLARE OUT_BAD            '/DIY/200/UC2/lineitem_errors'

-- Option to use replicated JOIN for the supplier name lookup.
-- use it
%DECLARE USE_REP_JOIN 'USING \'REPLICATED\' ' ;
-- don't use it
-- %DECLARE USE_REP_JOIN ' ' ;

-- REGEX to match valid Gregorian dates in the form yyyy-mm-dd for
-- years 0000 to 9999 (as a mathematical concept, not political).
%DECLARE VALID_DATE '^((?:\\d{4})-(?:0[1-9]|1[0-2])-(?:0[1-9]|1\\d|2[0-8])|(?0[13-
9]|1[0-2])-(?:29|30)|(?0[13578]|1[02])-(
31)|(?:\\d{2}(?:[02468][48]|[13579][26]|[2468]0)|(?:[02468][048]|[13579][26])00)-02-
29)$'

-- header for invalid data error message field and other nerror messages
%DECLARE ERR_MSG_HEAD 'ERROR(S) in LINEITEM source record:'
%DECLARE ORD_ERR ' invalid L_ORDERKEY value;'
%DECLARE PAR_ERR ' invalid L_PARTKEY value;'
%DECLARE SUP_ERR ' invalid L_SUPPKEY value;'
%DECLARE LIN_ERR ' invalid L_LINENUMBER value;'
%DECLARE QUN_ERR ' invalid L_QUANTITY value;'
%DECLARE DIS_ERR ' invalid L_DISCOUNT value;'
%DECLARE TAX_ERR ' invalid L_TAX value;'
%DECLARE RET_ERR ' L_RETURNFLAG must be one of A,N,R;'
%DECLARE LIS_ERR ' L_LINESTATUS must be one of O,F;'
%DECLARE SHI_ERR ' L_SHIPMODE must be one of AIR,FOB,MAIL,RAIL,REG AIR,SHIP,TRUCK;'
%DECLARE NAM_ERR ' supplier lookup failed;'
%DECLARE COD_ERR ' invalid L_COMMITDATE;'
%DECLARE SHD_ERR ' invalid L_SHIPDATE;'
%DECLARE RED_ERR ' invalid L_RECEIPTDATE;'
%DECLARE SRD_ERR ' L_SHIPDATE after L_RECEIPTDATE;'

-----
-----
-- start of the main program

-- data to be validated

lineitem = LOAD '$IN_LINEITEM_VENDOR' USING PigStorage('|') AS (
  l_orderkey:int, l_partkey:int, l_suppkey:int, l_linenummer:int, l_quantity:int,
  l_extendedprice:float, l_discount:float, l_tax:float,
  l_returnflag:chararray, l_linestatus:chararray,
```

```

    l_shipdate:chararray, l_commitdate:chararray, l_receiptdate:chararray,
    l_shipinstruct:chararray, l_shipmode:chararray,
    l_comment);

-- lookup table for supplier name and validation
-- only first two columns from supplier

suppname = LOAD '$IN_SUPPLIER' USING PigStorage('|') AS (
    s_suppkey:int, s_name:chararray);

-- anti join idiom (completed below) for the supplier lookup (i.e., null is if unmatched)
-- We use known cardinality of TPC-H tables (supplier is small; lineitem is
-- huge) to choose JOIN type. For example, when the TPC-H SF is 1000, the entire
-- supplier table is about 1.4GB and 10,000,000 rows. So, we know the supplier table
-- with two fields will fit (partitioned or not) in RAM -- so replicated JOIN

good1 = JOIN lineitem BY l_suppkey LEFT OUTER, suppname BY s_suppkey $USE_REP_JOIN ;

good2 = FOREACH good1 GENERATE
--
-- start of error message
    CONCAT('$ERR_MSG_HEAD',
    (l_orderkey > 0 ? '' : '$ORD_ERR'), CONCAT(
    (l_partkey > 0 ? '' : '$PAR_ERR'), CONCAT(
    (l_suppkey > 0 ? '' : '$SUP_ERR'), CONCAT(
    (l_linenum > 0 ? '' : '$LIN_ERR'), CONCAT(
    (l_quantity > 0 ? '' : '$QUN_ERR'), CONCAT(
    ((l_discount >= 0.0F AND l_discount <= 1.0F)
    ? '' : '$DIS_ERR'), CONCAT(
    (l_tax >= 0.0F ? '' : '$TAX_ERR'), CONCAT(
    (l_returnflag IN ('A','N','R')
    ? '' : '$RET_ERR'), CONCAT(
    (l_linestatus IN ('F','O')
    ? '' : '$LIS_ERR'), CONCAT(
    (l_shipmode IN ('AIR','FOB','MAIL','RAIL','REG AIR','SHIP','TRUCK')
    ? '' : '$SHI_ERR'), CONCAT(
    (s_name is NOT NULL ? '' : '$NAM_ERR'), CONCAT(
    (l_commitdate matches '$VALID_DATE'
    ? '' : '$COD_ERR'), CONCAT(
    (l_shipdate matches '$VALID_DATE'
    ? '' : '$SHD_ERR'), CONCAT(
    (l_receiptdate matches '$VALID_DATE'
    ? '' : '$RED_ERR'), -- last CONCAT
    ((l_shipdate matches '$VALID_DATE' AND l_receiptdate matches '$VALID_DATE')
    ? ((DaysBetween(ToDate(l_receiptdate,'yyyy-MM-
dd'),ToDate(l_shipdate,'yyyy-MM-dd')) >= 0)
    ? '' : '$SRD_ERR' )
    : ''))
    ))))))) AS err_message,
-- end of error message
--
    l_orderkey .. l_suppkey, s_name, l_linenum .. l_shipmode,
--
-- start of shipping time and rating as a TUPLE
    ((l_shipdate matches '$VALID_DATE' AND l_receiptdate matches '$VALID_DATE')
    ? TOTUPLE((int)DaysBetween(ToDate(l_receiptdate,'yyyy-MM-
dd'),ToDate(l_shipdate,'yyyy-MM-dd')),

```

```

                (CASE (int)DaysBetween(ToDate(l_receiptdate, 'yyyy-MM-
dd'),ToDate(l_shipdate, 'yyyy-MM-dd'))
                WHEN 0 THEN 'A' WHEN 1 THEN 'A' WHEN 2 THEN 'A'
WHEN 3 THEN 'A'
                WHEN 4 THEN 'B' WHEN 5 THEN 'B' WHEN 6 THEN 'B'
WHEN 7 THEN 'B'
                WHEN 8 THEN 'C' WHEN 9 THEN 'C' WHEN 10 THEN 'C'
WHEN 11 THEN 'C' WHEN 12 THEN 'C' WHEN 13 THEN 'C'
                ELSE 'D' END ) )
                : TOTUPLE( 999, 'Z')) AS t1:tuple(shippingtime_days:int,
shipping_ratingi:chararray),
-- end of shipping time and rating as a TUPLE
--
l_comment;

SPLIT good2 INTO
  good IF ENDSWITH(err_message, ':'),
  bad OTHERWISE;

good = FOREACH good GENERATE l_orderkey .. l_shipmode, FLATTEN(t1), l_comment;
bad = FOREACH bad GENERATE err_message .. l_suppkey, l_linenummer .. l_shipmode,
l_comment;

STORE good into '$OUT_GOOD' USING PigStorage('|');
STORE bad into '$OUT_BAD' USING PigStorage('|');

```

Use case 3: Vendor mainframe file integration

```
set pig.maxCombinedSplitSize 2147483648
set pig.exec.mapPartAgg true
```

```
-- uc3
-- djs, 13 july 2015
```

```
-----
-----
-- parameters and constants
```

```
-- input and output files
%DECLARE IN_ORDERITEM '/UC3/200/Source/ORDRITEM'
%DECLARE OUT_ORDERS   '/DIY/200/UC3/orders'
%DECLARE OUT_LINEITEM '/DIY/200/UC3/lineitem'
```

```
-----
-----
-- job parameters
-- set the property for fixed-lengthed records; unsure which it is
set mapreduce.input.fixedlengthinputformat.record.length 178;
set mapreduce.lib.input.fixedlengthinputformat.record.length 178;
set fixedlengthinputformat.record.length 178;
```

```
-- UDF loader: fixed-length reads and ebscdic decoing
register myudfs.jar;
```

```
-----
-----
-- this UDF loader reads fixed-length records (178 bytes), and performs
-- EBCDIC conversion on the string and formats the fields, using the fixed
-- copybook format for this table. A loader that works for
-- general-purpose copybooks is out of scope, but could be done with the
-- Jrecord classes.
```

```
cobol = LOAD '$IN_ORDERITEM' USING myudfs.SimpleFixedLengthLoader;
```

```
SPLIT cobol INTO
  orders   if (chararray)$0 == 'O',
  lineitem if (chararray)$0 == 'L';
```

```
-- remove record-type field
orders   = FOREACH orders   GENERATE $1 .. ;
lineitem = FOREACH lineitem GENERATE $1 .. ;
```

```
store orders   into '$OUT_ORDERS' USING PigStorage('|');
store lineitem into '$OUT_LINEITEM' USING PigStorage('|');
```

```
SimpleFixedLengthLoader.java
```

```
package myudfs;
/* compile with, for example,
  CP=/opt/cloudera/parcels/CDH-5.4.3-1.cdh5.4.3.p0.6/lib/pig/pig-0.12.0-cdh5.4.3.jar
  export PATH=$PATH:/usr/java/jdk1.7.0_67-cloudera/bin
```

```

export JAVA_HOME=/usr/java/jdk1.7.0_67-cloudera
javac -cp $CP SimpleFixedLengthLoader.java
*/
import java.io.IOException;
import java.util.List;
import java.util.ArrayList;
import java.util.Arrays;

import org.apache.hadoop.mapreduce.InputFormat;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.FixedLengthInputFormat;
import org.apache.hadoop.io.BytesWritable;
import org.apache.pig.FileInputLoadFunc;
import org.apache.pig.LoadFunc;
import org.apache.pig.PigException;
import org.apache.pig.backend.executionengine.ExecException;
import org.apache.pig.backend.hadoop.executionengine.mapReduceLayer.PigSplit;
import org.apache.pig.data.DataByteArray;
import org.apache.pig.data.Tuple;
import org.apache.pig.data.TupleFactory;
import org.apache.hadoop.classification.InterfaceAudience;

public class SimpleFixedLengthLoader extends LoadFunc {
    protected RecordReader in = null;
    private ArrayList<Object> mTuple = null;
    private TupleFactory mTupleFactory = TupleFactory.getInstance();

    public SimpleFixedLengthLoader() {
    }

    @Override
    public Tuple getNext()
    throws IOException {
        mTuple = new ArrayList<Object>();

        try {
            if (!in.nextKeyValue()) {
                return null;
            }
            BytesWritable val = (BytesWritable) in.getCurrentValue();
            byte[] buf = val.getBytes();
            String str = new String(buf, 0, 178, "Cp037");

            addTupleValue(mTuple, doSingle(str, 0)); // record type
            switch ( str.charAt(0) ) {
                case 'O':
                    addTupleValue(mTuple, cleanL(str, 1, 13)); // orderkey
                    addTupleValue(mTuple, cleanL(str, 13, 25)); // custkey
                    addTupleValue(mTuple, doSingle(str, 25)); // orderstatus
                    addTupleValue(mTuple, unpackBCD(buf, 26, 34)); // totalprice
                    addTupleValue(mTuple, doDate(str, 34)); // orderdate
                    addTupleValue(mTuple, cleanR(str, 42, 57)); // orderpriority
                    addTupleValue(mTuple, cleanR(str, 57, 72)); // clerk
                    addTupleValue(mTuple, doSingle(str, 72)); // shippriority
                    addTupleValue(mTuple, cleanR(str, 73, 152)); // comment
            }
        }
    }
}

```

```

        break;
    case 'L':
        addTupleValue(mTuple, cleanL(str, 1, 13)); // orderkey
        addTupleValue(mTuple, cleanL(str, 13, 25)); // partkey
        addTupleValue(mTuple, cleanL(str, 25, 37)); // suppkey
        addTupleValue(mTuple, cleanL(str, 37, 41)); // linenumber
        addTupleValue(mTuple, unpackBCD(buf, 41, 49)); // quantity
        addTupleValue(mTuple, unpackBCD(buf, 49, 57)); // extendedprice
        addTupleValue(mTuple, unpackBCD(buf, 57, 65)); // discount
        addTupleValue(mTuple, unpackBCD(buf, 65, 73)); // tax
        addTupleValue(mTuple, doSingle(str, 73)); // returnflag
        addTupleValue(mTuple, doSingle(str, 74)); // linestatus
        addTupleValue(mTuple, doDate(str,75)); // shipdate
        addTupleValue(mTuple, doDate(str,83)); // commitdate
        addTupleValue(mTuple, doDate(str,91)); // receiptdate
        addTupleValue(mTuple, cleanR(str, 99, 124)); // shipinstruct
        addTupleValue(mTuple, cleanR(str,124, 134)); // shipmode
        addTupleValue(mTuple, cleanR(str,134, 178)); // comment
        break;
    default:
        String unr = "Error: unknown record type";
        addTupleValue(mTuple, unr); // error message
        break;
}

Tuple tt = mTupleFactory.newTupleNoCopy(mTuple);
return tt;

} catch (InterruptedException e) {
    int errCode = 6018;
    String errMsg = "Error while reading input";
    throw new ExecException(errMsg, errCode,
        PigException.REMOTE_ENVIRONMENT, e);
}

}

private String unpackBCD(byte[] buf, int start, int end) {
    StringBuffer sb = new StringBuffer();
    byte bcd, high, low;
    for (int i = start; i < end-1; i++) {
        bcd = buf[i];
        high = (byte) (bcd & 0xf0);
        high >>>= (byte) 4;
        high = (byte) (high & 0x0f);
        low = (byte) (bcd & 0x0f);

        sb.append(high);
        sb.append(low);
    }

    bcd = buf[end-1];
    high = (byte) (bcd & 0xf0);
    high >>>= (byte) 4;
    high = (byte) (high & 0x0f);
    low = (byte) (bcd & 0x0f);
    sb.append(high);
    // add decimal -- no check for length
    sb.insert(sb.length()-2, '.');
}

```

```

// add sign
    if ( low == 0x0d ) sb.insert(0, '-');
    return (sb.toString()).replaceFirst("^(-?)0+(?!\\.)", "$1");
}
private String doSingle(String str, int start) {
    return str.substring(start, start+1);
}
private String cleanR(String str, int start, int end) {
    return str.substring(start, end).replaceFirst("\\s+$", "");
}
private String cleanL(String str, int start, int end) {
    return str.substring(start, end).replaceFirst("^0+(?!$)", "");
}
private String doDate(String str, int start) {
    return str.substring(start, start+4) + '-' + str.substring(start+4, start+6) +
    '-' + str.substring(start+6, start+8);
}
private void addTupleValue(ArrayList<Object> tuple, String buf) {
    tuple.add(new DataByteArray(buf));
}

@Override
public InputFormat getInputFormat() {
    return new FixedLengthInputFormat();
}

@Override
public void prepareToRead(RecordReader reader, PigSplit split) {
    in = reader;
}

@Override
public void setLocation(String location, Job job)
throws IOException {
    FileInputFormat.setInputPaths(job, location);
}
}

```

ABOUT PRINCIPLED TECHNOLOGIES



Principled Technologies, Inc.
1007 Slater Road, Suite 300
Durham, NC, 27703
www.principledtechnologies.com

We provide industry-leading technology assessment and fact-based marketing services. We bring to every assignment extensive experience with and expertise in all aspects of technology testing and analysis, from researching new technologies, to developing new methodologies, to testing with existing and new tools.

When the assessment is complete, we know how to present the results to a broad range of target audiences. We provide our clients with the materials they need, from market-focused data to use in their own collateral to custom sales aids, such as test reports, performance assessments, and white papers. Every document reflects the results of our trusted independent analysis.

We provide customized services that focus on our clients' individual requirements. Whether the technology involves hardware, software, Web sites, or services, we offer the experience, expertise, and tools to help our clients assess how it will fare against its competition, its performance, its market readiness, and its quality and reliability.

Our founders, Mark L. Van Name and Bill Catchings, have worked together in technology assessment for over 20 years. As journalists, they published over a thousand articles on a wide array of technology subjects. They created and led the Ziff-Davis Benchmark Operation, which developed such industry-standard benchmarks as Ziff Davis Media's Winstone and WebBench. They founded and led eTesting Labs, and after the acquisition of that company by Lionbridge Technologies were the head and CTO of VeriTest.

Principled Technologies is a registered trademark of Principled Technologies, Inc.
All other product names are the trademarks of their respective owners.

Disclaimer of Warranties; Limitation of Liability:

PRINCIPLED TECHNOLOGIES, INC. HAS MADE REASONABLE EFFORTS TO ENSURE THE ACCURACY AND VALIDITY OF ITS TESTING, HOWEVER, PRINCIPLED TECHNOLOGIES, INC. SPECIFICALLY DISCLAIMS ANY WARRANTY, EXPRESSED OR IMPLIED, RELATING TO THE TEST RESULTS AND ANALYSIS, THEIR ACCURACY, COMPLETENESS OR QUALITY, INCLUDING ANY IMPLIED WARRANTY OF FITNESS FOR ANY PARTICULAR PURPOSE. ALL PERSONS OR ENTITIES RELYING ON THE RESULTS OF ANY TESTING DO SO AT THEIR OWN RISK, AND AGREE THAT PRINCIPLED TECHNOLOGIES, INC., ITS EMPLOYEES AND ITS SUBCONTRACTORS SHALL HAVE NO LIABILITY WHATSOEVER FROM ANY CLAIM OF LOSS OR DAMAGE ON ACCOUNT OF ANY ALLEGED ERROR OR DEFECT IN ANY TESTING PROCEDURE OR RESULT.

IN NO EVENT SHALL PRINCIPLED TECHNOLOGIES, INC. BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES IN CONNECTION WITH ITS TESTING, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL PRINCIPLED TECHNOLOGIES, INC.'S LIABILITY, INCLUDING FOR DIRECT DAMAGES, EXCEED THE AMOUNTS PAID IN CONNECTION WITH PRINCIPLED TECHNOLOGIES, INC.'S TESTING. CUSTOMER'S SOLE AND EXCLUSIVE REMEDIES ARE AS SET FORTH HEREIN.
