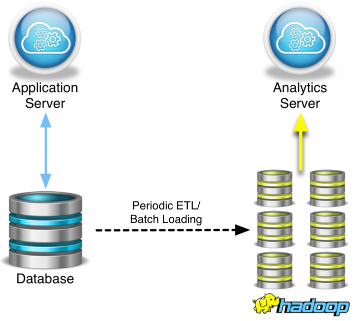
Continuent Tungsten

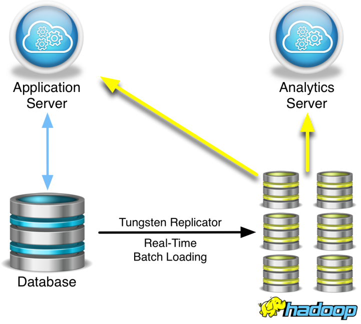
Real-Time Data Loading Into Hadoop

continuent

Table of Contents

Overview





Big data and the use of big data solutions for the processing and analysis of customer and logging data is increasing at a substantial rate. But whereas previously the analytical processing of data was handled in an offline and often separate part of the data processing process, Hadoop is now seen as an active part of the data flow. Rather than offline analytics, Hadoop is playing an active role in the processing and provisioning of data directly to the customer.

This presents a number of issues within the data exchange and migration process when using existing toolsets. They rely on a highly manual, periodic and intermittent transfer of information often closely tied to the previously non-interactive nature of the data transfer process. For example, a periodic ETL workflow in the illustration on the right.

Because of this, data needs to be loaded into Hadoop not in intermittent batches, but as an active database that sits alongside existing relational, NoSQL, and other databases. This allows data to be actively and constantly exchanged between the different database environments, ensuring consistency of the shared information. For example, using Tungsten Replicator Real-Time Batch Loading, analytics data becomes available for use shortly after it is written to the source database – see illustration on the right.

Existing Migration Technologies

There are a number of existing technologies that enable the dumping and loading of information between MySQL and Hadoop. These different technologies apply different techniques to achieve the loading process, and have different levels of compatibility and effectiveness, depending on the use case, data use, and required update rate.

A number of existing data migration technologies are available:

- SQL Dump and Import – This process relies on using existing database tools or custom scripts to extract and load data.
- Sqoop – A direct loading tool that exchanges information directly between existing databases and Hadoop by performing SELECT and export techniques, automating the core elements of the SQL dump and import approach.
- Talend Open Studio – A managed solution which makes use of the Sqoop connector, Pig, direct HDFS writes, and other standard Hadoop components to build a simplified extraction and migration interface using the Eclipse environment as the front end. Although it simplifies the process by making it accessible to end-users, Talend Open Studio has the same limitations as the base Sqoop tool.
- Quest SharePlex – A commercial tool that provides replication of data from the Oracle archive logs into Hadoop in near-real-time.
- Oracle GoldenGate – GoldenGate does not yet support direct loading and transfer of information from Oracle into Hadoop. Plans for introducing this functionality have been publicized. It will require significant configuration within the Pump process to provide the data migration and application into Hadoop from the Capture stream.

Tungsten Replicator provides an alternative to these solutions based on an existing, well-tested replication technology already used for both homogenous and heterogeneous replication between MySQL, Oracle, MongoDB, Vertica and other database solutions.

SQL Dump and Import

Advantages	Disadvantages
Flexible data selection	Incremental extraction requires database or application-level automation
Custom formatting and extraction	Requires additional database query overhead for export
Does not require replication to be enabled on the MySQL source	Requires manual transfer
	Entire database or table requires iterative process to read all data
	Requires read locks to ensure consistent data snapshots
	Does not capture full change history, only the difference between queries at the point of export

The simplest and most straightforward of the processes is to dump the information from the database into a CSV file and then manually load that into Hadoop. Although this process can be scripted, it can still be considered a manual, rather than automated, process. Even automated, the process relies on performing a manual export by directly querying the database.

The basic methodology for exporting data in this way is as follows:

Create a Comma Separated Value (CSV) file directly from within the source database environment. For example, within MySQL this can be achieved by running a SELECT … INTO OUTFILE statement, which generates a text file using the specified field and record delimiters.

The statement:

```
SELECT title, subtitle, servings, description into OUTFILE 'result.csv' FIELDS TERMINATED BY ','
FROM recipes
```

creates a file, result.csv, using the CSV format, selecting the specified columns into the generated file.

The query can be simple, as shown above, or complex, for example involving joins and selecting specific fields, columns and formatting as necessary, but the result is always a flat, non-relational copy of the source data.

For more complex interchange file generation, a custom script or application can be written to reformat the information into the desired format. This method is frequently used where the data needs further manipulation, or when data must be constructed into JavaScript Object Notation (JSON) or other formats that are not supported natively by the database as an export format.

Regardless of the method for generating the data content, the resulting file can then be loaded into Hadoop by using the hdfs command-line tool to copy the CSV file from the standard filesystem into HDFS:

```
$ hdfs dfs mkdir recipes
$ hdfs dfs -copyFromLocal recipes.csv recipes
```

Once the file has been imported into Hadoop, the data can be accessed and manipulated, for example, by creating a suitable Hive table definition that accesses the file directly.

The manual export/construction process has some advantages, in that it enables the maximum level of freedom in terms of the content and structure. However, the process must be completed in its entirety each time the data needs to be updated and extracted. Incremental changes are impossible without changing the underlying table structure to be able to determine when changes were made to the corresponding tables.

Sqoop

Advantages	Disadvantages
Flexible data selection	Incremental extraction requires database or application-level automation
Automates transfer	Requires additional database overhead for query-based export
Simplified multiple table extraction	Does not use explicit locking for data consistency
Does not require replication to be enabled on the MySQL source	Requires explicit schema and if necessary table selection; entire databases with multiple schemas must be selectively extracted
	Does not capture full change history, but only the differences between tables at the point of import

Sqoop is an open source tool that is provided as part of the standard Hadoop distribution. Sqoop uses existing JDBC drivers to communicate and execute queries on existing databases, including Oracle and MySQL, and then writes this table data directly into HDFS for use by the rest of the Hadoop system. CSV and other formats are supported. As part of Hadoop, it is open source and therefore easily audited and expanded.

Unlike the manual export and load process, Sqoop performs the entire export, format, and import process for you. Entire databases or selected tables can be extracted from the source database and written into Hadoop, and limited exchange of the information is provided in the opposite direction, from Hadoop back into the RDBMS.

The major advantage of Sqoop is that it effectively solves the manual dump and export solution by automating it as much as possible. The disadvantage is that Sqoop expects to transfer data into the database in CSV format and the structure cannot be altered.

Using Sqoop requires supplying the JDBC connection string, including the database that needs to be imported. In this case, all the tables from the specified database will be imported in one operation:

```
$ sqoop import-all-tables --connect jdbc:mysql://192.168.0.240/recipes
```

Sqoop performs the operation by running a standard SELECT statement on the source table and then translating the raw row-data into CSV format, before writing this information directly into HDFS. Tables are automatically partitioned into multiple files as they are written to HDFS, allowing for larger tables to be effectively distributed across the Hadoop cluster.

Using an additional direct option, data is extracted and written concurrently to multiple nodes in a given Hadoop cluster, increasing the speed of the transfer process.

Although Sqoop automates much of the basic transfer process, it is limited to extracting the data from the source RDBMS using SELECT operations, which makes the extraction of information an expensive process, particularly on very large tables. Whole table scans are always expensive within any RDBMS environment, as they require increased CPU and disk I/O to be processed.

Incremental extraction is possible with Sqoop, but only through a modification of both the database structure and application to enable changed rows of data to be identified so they can then be re-extracted and transferred to Hadoop. Two solutions are available: adding a lastmodified column, which is then used as the reference point for further extraction; or, using an append column which is used by the application to identify when an update or insert procedure has taken place. Sqoop can also be configured to automate the incremental process by creating a 'job' that contains the last extracted data information during an import:

```
$ sqoop job create nextimport --incremental append \
   --check-column id --last-value 4336573 \
   --connect jdbc:mysql://192.168.0.240/recipes \
   --username root --table recipes
```

This creates a specification in the file 'nextimport' that can be executed during the next update using:

```
$ sqoop job --exec nextimport
```

Sqoop simplifies much of the basic extraction and loading process, but achieves this by performing client-application level table queries, requiring additional database query load and without adding any benefits over the basic SQL dump and load process.

Talend Open Studio

Advantages	Disadvantages
Intermittent replication of fixed datasets	Incremental is supported in limited methods
Wide range of Hadoop interface targets such as Sqoop, Pig and HBase	Replication is not transfer safe, and recovery requires reload, rather than restart from position
Provides complex Export, Transform, Load (ETL) support	Heavy configuration required to identify source and target formats
Open Source license	
Automates transfer	

Talend Open Studio provides a GUI and suite of additional command-line tools that enable complex data migration and integration so that information can be moved from one database to another.

Using Eclipse as a front-end, Talend enables you to graphically draw data and information flows between sources and destinations. The actual loading process from MySQL or Oracle into Hadoop can be configured to make use of a wide variety of connectors, including using Sqoop, using custom direct HDFS writes, or using an interface through the Pig data transformation system within Hadoop to write and manage the flow of information.

Talend provides transfer of information by reading data from the existing database, or by using the Change Data Capture (CDC) model to extract data from the source database and then apply it. (The basic model for data migration is with fixed export from MySQL and insert into the target system for processing.)

Talend is not designed for long-term, permanent replication of data, but single exports, and periodic or regular exports of the information. Although it provides flexible output formats and translation or morphing of the data stream during the transfer process, each one is targeted at a specific existing dataset.

Talend is a flexible and capable product, but can be complex and long-winded to use, and it relies on the single, repeated, transfer of data, rather than a constant out of band direct replication of the information. This limitation also means that recovery and restarting from errors is more complex.

Quest SharePlex

Advantages	Disadvantages
Near-live data replication	Closed source, commercial product
Incremental data migration	Replication stalls and errors difficult to diagnose and recover from
	Requires separate configuration to select schemas and tables

Quest SharePlex is a commercial product that provides Oracle-to-Oracle replication and also provides support for replicating into Hadoop. SharePlex uses the Oracle redo logs (and archive logs) to generate the change information that is then replicated to the target database.

SharePlex uses a simple series of replication specifications that define the source and target databases and tables that provide the routing information between instances. Setup and configuration is complex and requires a number of different systems and steps, but the process is flexible and enables complex replication topologies to be supported.

Because the change information is read directly from the redo logs generated by Oracle, SharePlex automatically captures and replicates the stream of change data. This eliminates query-based extraction, and provides a near-live replication of the change information. Near-live replication is supported as the redo logs are periodically updated and checked for changes before the information is placed into a message queuing system that transfers and ultimately posts the changes into the target database.

Due to the constant stream of data changes generated, SharePlex includes support for identifying errors, but it is possible for different table transfers to be out of sync, and potentially to stop entirely, while other tables continue to receive changes and updates. This can allow relational databases to become out of sync, and block commits across multiple tables may fail or provide inconsistent data across a tableset. In addition, identifying the failed SQL statement requires examination of a single SQL log file of the changes.

Quest SharePlex provides some interesting functionality, but is both a commercial product and does not provide replication from MySQL.

Continuent Tungsten Replicator

Tungsten Replicator, released under a GPL V2 license, is an established solution that provides an alternative to native MySQL replication. Tungsten Replicator uses the MySQL binary log to extract and identify data changes within the core MySQL database. Tungsten Replicator takes this information, places the content into a Tungsten History Log and provides a single, identifiable, sequence number (transaction ID) that enables the state of replication to be easily identified. The sequence number also provides a convenient start/stop reference.

The core of Tungsten Replicator relies on two key services, the extractor, which extracts data from a supported database (MySQL, Oracle), and the applier, which writes data into a supported database (e.g. MySQL, Oracle, MongoDB). The data transfer between the two systems is based on the THL format, and the flexible information and structure that it supports. The two sides of the system work together to provide replication between different databases, even though the underlying database technologies may be different and have different functionality and data transaction expectations.

The Hadoop applier operates within this model, taking the raw THL data, and applying it into Hadoop, and then processing and merging the change information within Hadoop. Because the information is taken from the live stream of changes in the binary log, the replication of data is in real-time, with changes generated within MySQL transferred instantly through the binary log and THL to the applied database server. Transactions written to the binary log are transaction safe, and this safety is replicated in sequence to the target database, without fear of corruption or inconsistency.

How the Hadoop Applier operates

The Hadoop applier within Tungsten Replicator operates by taking row-based THL data from a Tungsten Replicator master and writing a CSV file of the changes. The production and content of the CSV file is significant.

The file is generated by batching a group of changes into a single CSV file. The resulting CSV file is then loaded into HDFS using a JavaScript based processor that executes the necessary commands within Hadoop to apply the files into your Hadoop cluster. Batching enables the volume of information being captured to be maximized into large files. These files are more efficient when loading the data into Hadoop because they maximize the distributed nature of the Hadoop Disk File System (HDFS). The number of transactions and batch intervals can be controlled to maximize the efficiency and load latency.

The application of the batch file loading into Hadoop is built on the core HDFS and Hadoop environments, and it is compatible with a range of different Hadoop flavors, including Cloudera, HortonWorks and IBM InfoSphere BigInsights. The append-only nature of HDFS means that direct replication of UPDATE and DELETE operations within the stream of database changes must be handled differently than within a normal RDBMS. The key to this process is how the information is generated within the CSV file.

The generated CSV file has a specific format that contains not just the row information, but also details of the update type, transaction sequence number, row_id within the batch, and the time of the original commit on the source server. Individual row data is tagged according to the operation type:

- An INSERT is recorded as a normal row insert.
- A DELETE is recorded as a deletion using the unique ID for that row from the source database.
- An UPDATE is recorded as a DELETE operation, followed by an INSERT of the new version of the data.

The generated CSV file contains one or more rows for each updated row entry, and a batch file may contain multiple entries for a single row as the row is updated multiple times. During replication into Hadoop, the unique, incremental, sequence number generated by each event when it is recorded in the THL can then be used to pick the 'latest' version of a row.

```
INSERT INTO messages (id, message) VALUES (10,’First Message’);
UPDATE messages SET message = ’Second Message’ WHERE id = 10;
INSERT INTO messages (id, message) VALUES (11,’Interim Message’);
UPDATE messages SET message = ’Third Message’ WHERE id = 10;
DELETE messages WHERE id = 11;
```

is represented within the CSV as:

Operation	Number	Primary key	ID column	Message column
I	1	10	10	First message
D	2	10	10	
I	2	10	10	Second message
I	3	11	11	Interim message
D	4	10	10	
I	4	10	10	Third message
D	5	11	11	

The information can be used to identify the final version by processing all deletes, but inserting only the latest sequence number of the information. In the above table, sequence 4 contains the 'current' version of the information. With a complete record of these changes, any point in the replication of information can be used as the 'current' maximum, simply by selecting the appropriate sequence number.

Within Hadoop, the final version of the information can be determined by performing a Map/Reduce operation, or, by using the SQL-like capabilities of Hive to handle the Map/Reduce job generation to generate the final version of the data. This can be achieved, either by looking entirely at the CSV data, or by merging existing row data with the change data. Tungsten Replicator can handle this merge and commit process automatically as part of the replication procedure, or it can be schedule through outside mechanisms such as Cron or Oozie.

Tungsten Replicator can work with the existing provisioned or transferred data, such as table data previously transferred by Sqoop, and combine this existing information with the incoming stream of changes that the replicator is generating and transferring into Hadoop. This allows Tungsten Replicator to be used with both existing and new installations with no difference in deployment or flexibility.

Replicator Master

The master side of the Tungsten Replicator reads the information from the source database. The example shown here operates from a MySQL host, although an Oracle master would also extract the information in a suitable format for replicating into Hadoop. With MySQL, data is extracted from the MySQL binary log by reading and parsing the binary log contents directly.

The MySQL binary log is automatically generated by MySQL and contains a transaction-aware and safe stream of all the changes made to the MySQL server. Because this information is generated automatically by MySQL, the content and validity of the information is assured. Furthermore, because this information is generated by the MySQL server and is accessible outside of the server, reading and using the content is a low-impact process. Data is not forcibly extracted from the server, but passively read from the binary log already used by native MySQL replication.

In addition, because the extractor is reading information from the binary log, the information is existing data created as a side-effect of transaction commits. Neither the application or schema need to be modified for the information to be extracted and written to the THL for use by the slave replicator.

The only requirement for deploying the replicator is that each table has a primary key, as this is required to effectively merge the data, and that MySQL has been configured to use row-based logging by setting the binlog_format variable globally to the ROW format.

To install the replicator, download the Tungsten Replicator package, extract the files, and then change into the created directory. Installation uses a tool called tpm, the Tungsten Package Manager; it automates the installation process copying the necessary files and setting the configuration.

The following command will install a Tungsten Replicator service reading from a local MySQL binary log to generate the THL information:

```
$ ./tools/tpm install alpha \
--install-directory=/opt/continuent \
--master=host1 \
--members=host1 \
--java-file-encoding=UTF8 \
--java-user-timezone=GMT \
--mysql-enable-enumtostring=true \
--mysql-enable-settostring=true \
--mysql-use-bytes-for-string=false \
--svc-extractor-filters=colnames,pkey \
--property=replicator.filter.pkey.addColumnsToDeletes=true \
--property=replicator.filter.pkey.addPkeyToInserts=true \
--replication-password=password \
--replication-user=tungsten \
--skip-validation-check=HostsFileCheck \
--skip-validation-check=ReplicationServicePipelines \
--start-and-report=true
```

The above configuration includes everything required, including the use of filters that provide column name and primary key information. Once installed, the replicator will start extracting information from the existing binary logs and writing the data into THL. The replicator also opens a network service so that the THL can be read remotely from the slave replicator that will apply those changes to Hadoop.

Replicator Slave

The slave replicator reads the information from the remote replicator service and writes the information into Hadoop in a number of stages:

1. Batches the data into suitably sized blocks (configurable).
2. Writes the CSV data, translating the updates into deletes and inserts, and formatting the file accordingly; a different file is created for each table and schema.
3. Copies the CSV file into HDFS.
4. Executes a Hive query to merge the batched CSV change data into the final tables.

The process and intervals of different steps of the operation are highly configurable.

For example, with a busy database where low-latency of the live data in Hadoop is not required, the batch file commit size can be set very high with a high interval. This allows for larger files and less-intermittent loads. Conversely, lower intervals and block commit sizes give more frequent loads, resulting in a lower latency between the MySQL master and the Hadoop slave.

To install the slave replicator, the same basic process is followed, using tpm to perform the installation that installs and copies the necessary files before starting the replicator.

```
$ ./tools/tpm install alpha \
--batch-enabled=true \
--batch-load-language=js \
--batch-load-template=hadoop \
--datasource-type=file \
--install-directory=/opt/continuent \
--java-file-encoding=UTF8 \
--java-user-timezone=GMT \
--master=host1 \
--members=host2 \
--property=replicator.datasource.applier.csvType=hive \
--property=replicator.stage.q-to-dbms.blockCommitInterval=1s \
--property=replicator.stage.q-to-dbms.blockCommitRowCount=1000 \
--replication-password=secret \
--replication-user=tungsten \
--skip-validation-check=DatasourceDBPort \
--skip-validation-check=DirectDatasourceDBPort \
--skip-validation-check=HostsFileCheck \
--skip-validation-check=InstallerMasterSlaveCheck \
--skip-validation-check=ReplicationServicePipelines \
--start-and-report=true
```

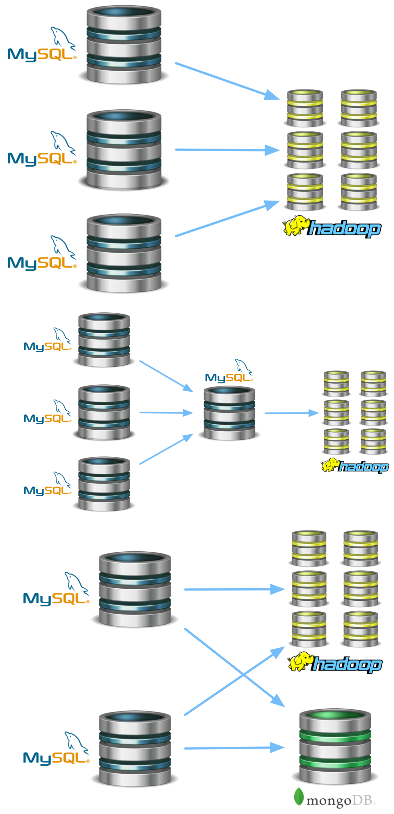
The hostname for the master defines the source for the THL and replicator information. The hive CSV sets conventions to produce data in a form preferred by Hive. The template type specifies the hadoop command to load the data. Using the hadoop command avoids complex dependencies on Hadoop Java libraries.

Once the replicator has been configured, the status of the replicator can be monitored by using the trepctl command. This provides detailed information about the current status, including the sequence number and latency of writes into Hadoop.

```
NAME                          VALUE
-------                       --------
appliedLastEventId     :       mysql-bin.000007:0000000008663481;-1
appliedLastSeqno       :      436
appliedLatency         :       0.0
channels               :       1
clusterName            :      alpha
currentEventId         :      NONE
currentTimeMillis      :      1391668558116
dataServerHost         :      192.168.1.252
extensions             :
host                   :      192.168.1.252
latestEpochNumber      :      0
masterConnectUri       :      thl://tr-hadoop1:2112/
masterListenUri        :      null
maximumStoredSeqNo     :      436
minimumStoredSeqNo     :      0
offlineRequests        :      NONE
pendingError           :      NONE
pendingErrorCode       :      NONE
pendingErrorEventId    :      NONE
pendingErrorSeqno      :      -1
pendingExceptionMessage :     NONE
pipelineSource         :      thl://tr-hadoop1:2112/
relativeLatency        :      -974267.884
resourcePrecedence     :      99
rmiPort                :      10002
role                   :      slave
seqnoType              :      java.lang.Long
serviceName            :      alpha
serviceType            :      local
simpleServiceName      :      alpha
siteName               :      default
sourceId               :      192.168.1.252
state                  :      ONLINE
timeInStateSeconds     :      2.006
transitioningTo        :
uptimeSeconds          :      96575.133
useSSLConnection       :      false
version                :      Tungsten Replicator 2.2.9 build 9
Finished status command...
```

Because the sequence number is unique, replication can be paused or stopped, and then later restarted without loss or corruption of the data stream. Because the data stream is sequential and transaction safe, and only complete transactions are transferred, the data is always consistent for each transaction.

Tungsten Replicator Topologies



Tungsten Replicator supports a number of different topologies, and these can be exploited when combined with the Hadoop applier to enable a range of different data replication structures.

A standard replication service could replicate data from multiple MySQL servers into a single Hadoop cluster, using Hadoop to provide the consolidation of information. This can be useful when you have multiple MySQL servers providing a shared data architecture, but want to perform analytics on the full body of data across all servers. For example, a fan-in topology with multiple sources applied to one Hadoop target.

Alternatively, a 'fan-in aggregation' topology enables data to be combined within a MySQL server before replication into Hadoop.

Different replicators and heterogeneous targets can also be combined, reading from the same Tungsten Replicator master, and applying to multiple different heterogeneous targets. For example, Tungsten Replicator could be configured to read information from the master and replicate some of this information into a MongoDB server for use by the application servers for document-based access to the content.

Simultaneously, the data could also be written into a Hadoop cluster for the purposes of analytics to analyze the entire database or change information.

Tungsten Replicator Summary

Tungsten Replicator supports a number of different topologies, and these can be exploited when combined with the Hadoop applier to enable a range of different data replication structures.

Advantages	Disadvantages
Near-live data replication	Existing data must be separately provisioned
Open source	Requires use of row replication on MySQL master
Automated transfer of the entire database, or selected databases and tables	
Incremental data migration, including start/stop support	
Automated transfer of the entire database, or selected databases and tables	
Low-load extraction of the source data; requires no access to the source database or query execution	
Requires no database or application changes	

About Continuent

Continuent is a leading provider of database clustering and replication, enabling enterprises to run business-critical applications on cost-effective open source software. Continuent Tungsten provides enterprise-class high availability, globally redundant data distribution and real-time heterogeneous data integration in cloud and on-premises environments. Our customers represent the most innovative and successful organizations in the world, handling billions of transactions daily across a wide range of industries.

For more information on our products and services, please visit www.continuent.com, email us at sales@continuent.com or call us at (800) 270-9035, and follow us on Twitter @Continuent.