

Integrating VoltDB with Hadoop

Hadoop is an open source framework for managing and manipulating massive volumes of data. VoltDB is a database for handling high velocity data. Business solutions often require both real-time data management and deep historical data analysis. The ability of VoltDB to export selected data "on the fly" lets you integrate VoltDB and Hadoop to address both needs. This whitepaper explains how to use export to integrate VoltDB and Hadoop within a larger information infrastructure.

VoltDB, Inc.

Introduction

VoltDB is the world's most performant RDBMS. It can out perform other commercial databases in transactions per second (the most common measure of throughput) by 30-40 times. VoltDB also scales easily, both in throughput and capacity, because of its distributed, shared-nothing architecture.

VoltDB is very effective at managing large, active datasets and the transactions that use them. However, business systems often need access to historical data — information from past transactions — for business intelligence and other forms of data mining. This sort of information can be both massive in volume and require complex queries to sort through. Hadoop excels at complex analytical operations on deep historical data sets.

Businesses that need both exceptional transactional throughput and query access to massive amounts of historical data need to integrate VoltDB with other technologies.

Hadoop is an open source framework for managing large datasets. The Hadoop framework includes both distributed processing (commonly known as map/reduce functions) and a distributed file system (hdfs). Because of its distributed architecture, Hadoop can handle massive volumes of data. The advantage of Hadoop is that it separates the physical storage of the data from the application interface for reading and writing. The client application does not need to know where the data is stored; Hadoop presents itself as a generic file system.

VoltDB Export

VoltDB lets you automate the export process by specifying certain tables in the schema as sources for export as part of the project definition file. At runtime, any data written to the specified tables is sent to the export *connector*, which manages the exchange of the updated information to a separate receiving application. Figure 1, “Overview of VoltDB Export Process” illustrates the basic structure of the export process, where Tables B and D are specified as export tables.

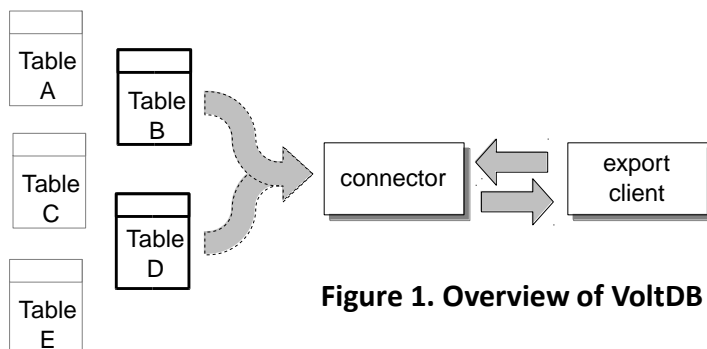


Figure 1. Overview of VoltDB Export Process

Note that you, as the application developer, do not need to modify the schema or the client application to turn exporting of live data on and off. The application's stored procedures insert data into the export-only tables; but it is the deployment file that determines whether export actually occurs at runtime.

When a stored procedure uses an SQL INSERT statement to write data into an export-only table, rather than storing that data in the database, it is handed off to the connector when the stored procedure successfully commits the transaction. Export-only tables have several important characteristics:

- ✔ Export-only tables let you limit the export to only the data that is required. For example, in the preceding example, Table B may contain a subset of columns from Table A. Whenever a new record is written to Table A, the corresponding columns can be written to Table B for export to the remote database.
- ✔ Export-only tables let you combine fields from several existing tables into a single exported table. This technique is particularly useful if your VoltDB database and the target of the export have different schema. The export-only table can act as a transformation of VoltDB data to a representation of the target schema.
- ✔ Export-only tables let you control *when* data is exported. Again, in the previous example, Table D might be an export-only table that is an exact replica of Table C. However, the records in Table C are updated frequently. The client application can choose to copy records from Table C to Table D only when all of the updates are completed and the data is finalized, significantly reducing the amount of data that must pass through the connector.

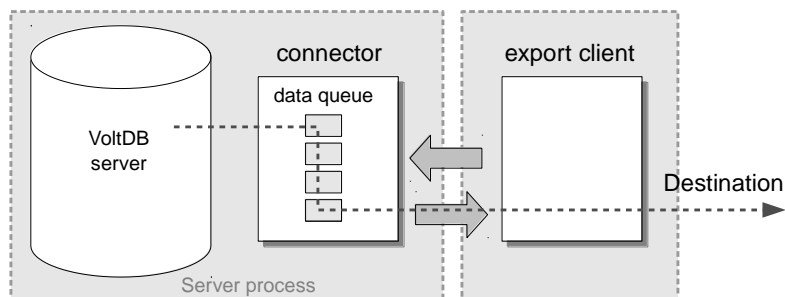
How Export Works

The export connector implements a loosely coupled approach to extracting export data from a running VoltDB database. When export is enabled at runtime:

1. Insert operations to the database tables identified as export-only in the project definition file are queued to the export connector.
2. An export client establishes a link to the connector through one of the standard TCP/IP ports (either the client or admin port). The client then issues POLL requests.
3. The connector responds to the POLL requests with the next queued data block (or an empty block if the queue is empty).
4. The client is then responsible for receiving the data and writing it to the appropriate destination.
5. Finally, the export client sends an ACK message acknowledging completion of the export (at which point the connector can remove it from the queue) before polling for the next data block.

Figure 2, “Components of the VoltDB Export Process” shows the interaction between the VoltDB database, the connector, and the export client.

Figure 2. Components of the VoltDB Export Process



The export function queues and passes data to the connector automatically. You do not need to do anything explicitly to start the connector; it starts and stops when the database starts and stops. The connector and the export client use a series of poll and ack requests to exchange the data over the TCP port.

The export client decides what is done with the data it receives from the connector. For Hadoop output, the export-to-Hadoop client uses the Sqoop importer from Cloudera to write the exported data to hdfs.

Export Overflow

For the export process to work, it is important that the connector and client keep up with the queue of exported information. If too much data gets queued to the connector by the export function without being fetched by the client, the VoltDB server process consumes increasingly large amounts of memory.

If the export client does not keep up with the connector and the data queue fills up, VoltDB starts writing overflow data in the export buffer to disk. This protects your database in several ways:

- ✓ If the client fails, writing to disk helps VoltDB avoid consuming too much memory while waiting for the client to restart.
- ✓ If the database is stopped, the export data is retained across sessions. When the database restarts and the client reconnects, the connector will retrieve the overflow data and reinsert it in the export queue.

You can specify where VoltDB writes the overflow export data using the `<exportoverflow>` element in the deployment file. For example:

```
<paths>
  <voltdbroot path="/opt/voltdb/" />
  <exportoverflow path="/tmp/export/" />
</paths>
```

If you do not specify a path for export overflow, VoltDB creates a subfolder in the root directory (in the preceding example, `/opt/voltdb`).

Persistence Across Database Sessions

It is important to note that VoltDB only uses the disk storage for overflow data. However, you can force VoltDB to write all queued export data to disk by either calling the `@Quiesce` system procedure or by requesting a blocking snapshot. (That is, calling `@SnapshotSave` with the blocking flag set.) This means it is possible to perform an orderly shutdown of a VoltDB database and ensure all data (including export data) is saved with the following procedure:

1. Put the database into admin mode with `@Pause`.
2. Perform a blocking snapshot with `@SnapShotSave`, saving both the database and any existing queued export data.
3. Shutdown the database with `@Shutdown`.

You can then restore the database — and any pending export queue data — by starting the database in admin mode, restoring the snapshot, and then exiting admin mode.

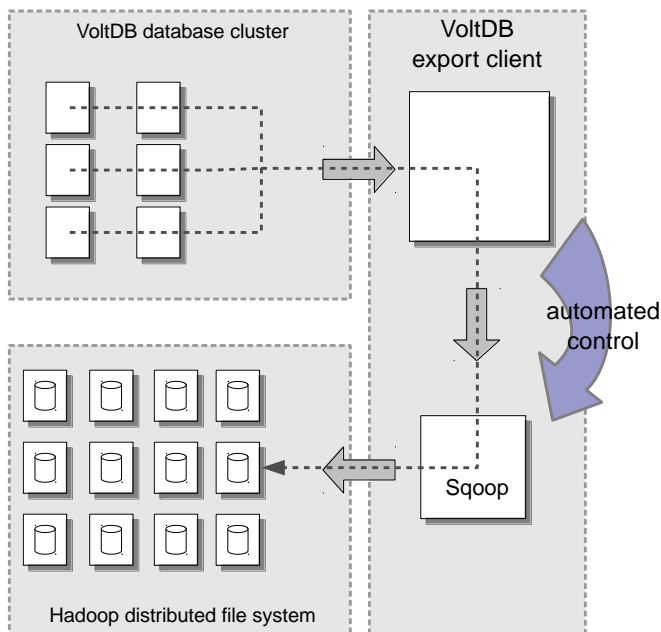
The VoltDB Export-to-Hadoop Client

VoltDB Enterprise Edition version 1.3.3 and later comes with an export client that fully automates the process of export and import between VoltDB and Hadoop. This end-to-end integration is possible due to the VoltDB export client integrating with the Sqoop SQL import technology developed by Cloudera.

When you start the export client, you specify the cluster nodes for the client to query for information (using the `--servers` argument). The client queries these nodes, one at a time, until it receives a response. Part of the response it receives is a description of the cluster, including a list of nodes and available ports. The client then creates connections to every node in the cluster.

Once the client connects to the cluster, it starts to poll and ack for export data. The client "decodes" the export stream from its serialized form into the appropriate datatypes for each column in the table. The VoltDB export client then uses Sqoop to read the data into the Hadoop distributed file system. Figure 3, "The VoltDB Export-to-Hadoop Process in Action" illustrates this process.

Figure 3. VoltDB Export-to-Hadoop Process in Action



If the export client loses connection to any of the nodes in the VoltDB cluster (either because of a node failure or a shutdown), it disconnects from the cluster and repeats the initial discovery process, using the information it collected from the original connection. Once the cluster comes back, the client resumes export operations, picking up with the last data packet it received prior to the interruption. This allows the export process to continue without operator intervention even across network disruptions, node failures, and database sessions.

How to Run the Export-to-Hadoop Client

To use the export-to-Hadoop client, you must have already installed and configured both Hadoop and Sqoop and started the Hadoop file system. The client uses the variable `HADOOP_HOME` to determine where Hadoop is installed and what file system to use as the target of the export. It then extracts and formats the exported data from the VoltDB connector and runs the Sqoop importer to read that data into Hadoop.

You start the export-to-Hadoop client using the Java command. However, it is important that all of the necessary JAR files for VoltDB, Hadoop, and Sqoop are in your class path. The easiest way to do this is using an Ant build script. But for demonstration purposes, the following example uses the export command to define CLASSPATH.

The command to start the export-to-Hadoop client looks something like the following:

```
$ V_PATH="/opt/voltdb/voltdb/*"
$ H_PATH="$HADOOP_HOME/*:$HADOOP_HOME/conf:$HADOOP_HOME/lib/*"
$ S_PATH="$SQOOP_HOME/*:$SQOOP_HOME/lib/*"
$ export CLASSPATH="$V_PATH:$H_PATH:$S_PATH"
$ java -Djava.library.path=/home/me/voltdb/voltdb/ \
    org.voltdb.hadoop.VoltDBSqoopExportClient \
    --connect client \
    --servers myserver \
    --nonce ExportData \
    --type csv
```

The Export-to-Hadoop Client Command Line

The export-to-Hadoop client has a number of command line options that let you customize the export process to meet your needs. Many of the options are the same as for the export-to-file client. However, some options are specific to this client and allow you to control the Sqoop importer. In the following description the generic export client options are listed separately from the Sqoop-specific options.

The complete syntax of the command line is as follows:

```
$ java -Djava.library.path={path} -classpath {path} \
    org.voltdb.hadoop.VoltDBSqoopExportClient \
    {arguments...}
$ java -Djava.library.path={path} -classpath {path} \
    org.voltdb.hadoop.VoltDBSqoopExportClient \
    --help
```

The supported export client arguments are:

`--servers {host-name[:port]} [,...]`

A comma separated list of host names or IP addresses to query.

`--nonce {text}`

The prefix to use for the directories that the client creates in Hadoop.

`--connect {client|admin}`

The port to connect to. You specify the type of port (client or admin), not the port number.

`--user {text}`

The username to use for authenticating to the VoltDB server(s). Required only if security is enabled for the database.

`--password {text}`

The password to use for authenticating to the VoltDB server(s). Required only if security is enabled for the database. If you specify a username but not a password, the export client prompts you for the password.

`--period {integer}`

(Optional.) The frequency, in minutes, for "rolling" the output file. The default frequency is 60 minutes.

`--outdir {path}`

(Optional.) The directory where temporary files are written as part of the export/import process.

`--delimiters {text}`

(Optional.) Alternate delimiter characters for the CSV output. The text string specifies four characters: the field delimiter, the record delimiter, the enclosing character and the escape character. To use special or non-printing characters (including the space character) encode the character as an html entity. For example "<" for the "less than" symbol.

`--require-enclosed-output`

(Optional.) Specifies that all CSV fields, even numeric and null fields, are enclosed (in quotation marks, by default).

`--http-port {integer}`

(Optional.) The http port for connecting to the Hadoop file system (port 8099 by default).

`--skipinternals`

(Optional.) Eliminates the six columns of VoltDB metadata (such as transaction ID and timestamp) from the output. If you specify `--skipinternals` the output files contain only the exported table data.

The following are supported Sqoop-specific arguments. These arguments are passed directly to the Sqoop importer interface and are documented in detail in the Sqoop documentation:

`--hadoop-home {path}`

(Optional.) Overrides the value of the `HADOOP_HOME` environment variable. You must provide a valid Hadoop home directory, either by defining `HADOOP_HOME` or specifying the location with `--hadoop-home`.

`--verbose`

(Optional.) Displays additional information during Sqoop processing.

`--target-dir {path}`

(Optional.) The destination directory in HDFS.

`--warehouse-dir {path}`

(Optional.) A parent directory in HDFS where separate folders are created for each table. The options `--target-dir` and `--warehouse-dir` are mutually exclusive.

`--null-string {text}`

(Optional.) The string to use for null string values in the output.

`--null-non-string {text}`

(Optional.) The string to use for null values in the output for all datatypes except strings.

Summary

Organizations increasingly need to ingest and analyze high velocity data in real-time, and then collect that data into a historical datastore for further analysis. VoltDB is an excellent solution for handling data in high velocity state; Hadoop is an excellent solution for analyzing massive volumes of historical data. Thus, the combination of VoltDB and Hadoop offer you the flexibility to handle a continuum of “fast” and “deep” data applications.

VoltDB Export-to-Hadoop provides you with the following capabilities and benefits:

The flexibility to select which data will be exported from VoltDB to Hadoop.

The ability to enrich data, using standard SQL and relational semantics, in VoltDB prior to exporting it to Hadoop.

The ability to quickly configure VoltDB with your Sqoop/Hadoop settings, minimizing the time and effort needed to bring your infrastructure online.

A robust, loosely-coupled product integration that allows the different components to operate at different levels of throughput, availability and processing state.