VOLTDB

HADOOP DATABASE INTEGRATION

Hadoop is an open-source framework for managing and manipulating massive amounts of data. VoltDB is an OLTP database for handling large volumes of transactions. Business solutions often require both instantaneous, reliable processing of transactions as well as archiving and analysis of historical data. 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 the larger business context.

INTRODUCTION

VoltDB is the world's most performant OLTP database. 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. Neither of which VoltDB is particularly well suited for.

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. Because of its distributed architecture, Hadoop can handle massive (in the order of terabytes and petabytes) 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.empowers businesses to quickly assess vital customer information that can mean the difference between making and losing thousands of dollars.

INTEGRATING VOLTDB WITH HADOOP

It is possible to design and develop a complete business solution utilizing both VoltDB and Hadoop from scratch. But you do not need to. VoltDB simplifies the process by providing an export facility that lets you automatically archive selected data from the VoltDB database. And you can use this export functionality with Hadoop.

As delivered, the VoltDB export feature comes with a very simple example export client, called the export-to-file receiver. The export-to-file receiver writes the exported data to a single file for each table. This makes it easy to see how the data is exported; it is instructive but not what is needed from a Hadoop perspective.

To make the data easier to manipulate, sort, and filter in Hadoop, you need to write the information out in smaller chunks. The Twitter example application (which is also part of the standard VoltDB software distribution) does this. The Twitter example provides a customized export client that writes out separate files for each table in the database schema, segmented by time to keep the files small.

Writing a custom export client is not difficult, since you can reuse the existing classes and interfaces from the export-tofile receiver. The following sections explain how VoltDB export works, how the Twitter example uses a custom export receiver to integrate with Hadoop, and how to create a custom export client of your own.

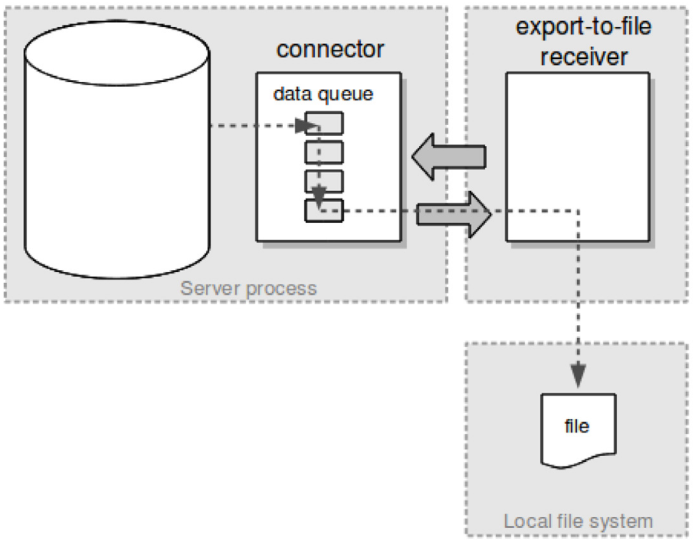
HOW EXPORT WORKS IN VOLTDB

As part of the VoltDB project definition you can define whether export is enabled and, if so, which tables are exported. Whenever those tables are inserted, updated, or deleted, a record of the transaction is written into the export buffer.

The export client (or receiver) communicates with the export connector function through a series of POLLs and ACKs to query and fetch the data in the export queue. The receiver is then responsible for writing that data to the appropriate external data store. (For details on how export works, see the chapter on "Exporting Live Data" in the VoltDB User's Guide.)

In the case of the export-to-file receiver, all of the export data is written to a single file per table in local storage, as shown in Figure 1, "Generic Export to File Receiver".

Figure 1. Generic Export to File Receiver



The generic export-to-file receiver is useful because it illustrates how export works and it generates a few comma-delimited files that can be used as import into another system.

CUSTOMIZING THE EXPORT RECEIVER TO WORK WITH HADOOP

However, a single large file per table is not very useful when trying to sort and search the exported data for specific information. It is easier if the exported data is segmented in some meaningful way that can be used for post processing.

This is exactly what the Twitter sample application does. Instead of using the default receiver, the Twitter sample creates a custom receiver that sorts the exported data into separate files, segmented both by database table and over time. More importantly, the custom receiver uses the Hadoop interface to write those files into the Hadoop file system (hdfs).

Note that you do not need to create a custom receiver from scratch; you can reuse many of the classes and interfaces of the default receiver to do the work of retrieving the data from the VoltDB export connector and interpreting the individual records.

In the Twitter example, the custom receiver is created by defining a class (ExportToHDFSClient) that extends the ExportClientBase class. This extended ExportToHDFSClient class performs the following functions:

1. Opens the first argument to the class as a Hadoop file system
2. Interprets the second argument as the list of VoltDB servers to connect to (these first two actions emulate the export-to-file client interface)
3. Overrides the ExportDecoderBase interface, adding a custom decoder to the list for every data source.
4. Calls connectToExportServers to connect to the database servers and begin processing the exported data.

Most of the ExportToHDFSClient class is just a template that creates the necessary context for establishing a client process and starting the export function. The main customization is initiating the Hadoop file system and replacing the list of export decoders with a custom decoder. Decoders are used to interpret and process each block of exported data as it is fetched from the VoltDB database servers.

For the Twitter example, the custom decoder is defined in the class ExportToHDFSDecoder, which extends Export-DecoderBase. Again, the custom decoder takes advantage of the base classes and methods defined for the default export client and overrides only those classes specific to exporting records for Hadoop.

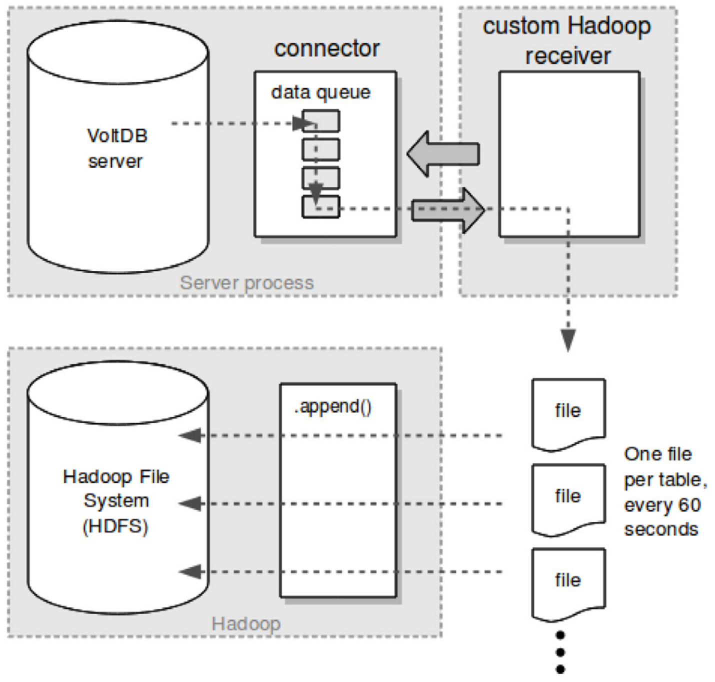
In this case, the custom decoder overrides the method processRow. The customization still doesn't need to actually interpret the incoming data; it uses the default decodeRow method to do that work. Instead, the customized processRow method writes the decoded row into a text string and then appends it to the output using a special class, RollingAppender.

RollingAppender is used instead of direct I/O to avoid creating files that are too big or too small. Too big and the resulting output can compromise the efficiency of Hadoop map/reduce functions. Too small and the overhead of creating millions of tiny files can swamp the export client and cause it to fall behind processing the stream of export data.

Rather than writing each record to a file, RollingAppender provides a simple output buffer, where files are written once every 60 seconds. It may be useful to reuse this technique when creating your own custom export clients.

Figure 2, "Custom Hadoop Receiver" illustrates how the Twitter sample application integrates VoltDB's export function with the Hadoop file system through relatively simple customizations to the default export receiver.

Figure 2. Custom Hadoop Receiver



The source code for all three components of the custom export receiver (ExportToHFDFSClient, ExportToHDFSDecoder, and RollingAppender) are available in the VoltDB sample applications folder under twitter/src/org/voltdb/twitter/hadoop/hdfs/.

Although it is not specific to VoltDB (since once the files are written into the Hadoop file system, the technical aspect of integrating VoltDB and Hadoop is complete), the Twitter sample also includes an example map/reduce function

to process the exported data. The JobRunner class (found in twitter/src/org/voltdb/twitter/hadoop/mr/) can be run as a Hadoop tool to process the data exported from VoltDB and summarize the number of instances of each hash tag. In most cases, a similar map/reduce function is needed to provide meaningful summaries of the data for business applications accessing the Hadoop archive.

CREATING YOUR OWN CUSTOM EXPORT RECEIVER

Each business case is unique, but the Twitter example demonstrates one way to integrate VoltDB with Hadoop. Depending upon the business specifics, your use of VoltDB and Hadoop will vary, but the overall process is the same :

1. Create a client class that extends ExportClientBase. Extending ExportClientBase ensures your client application has access to all of the methods and data structures needed to fetch and interpret the export data from VoltDB. The client class should also do the following:
    - Define any output ports and/or interfaces needed to write out the export data. In the case of integrating with Hadoop, this means opening the appropriate Hadoop file system.
    - Override the ExportDecoderBase. ExportDecoderBase is invoked every time a block of export data is retrieved from the VoltDB database. In the Twitter example, this overridden class is used to invoke separate instances of a ExportDecoderBase class depending on the table name. This is often a useful way of segmenting the export data. However, your business case will help you decide how to distinguish the different export records.
    - Invoke connectToExportServers specifying a list of all of the servers that make up the database cluster. (You must connect to all of the active servers to ensure you are collecting all of the export data.)
2. Define a class that extends the ExportDecoderBase. This is the class that is executed each time a block of export data is received by the export client. This is the class that is responsible for interpreting the data and writing it to an output destination in the appropriate format. You can write the ExportDecoderBase class from scratch if necessary, but in most cases it is easier to reuse the existing methods in the base class.

You will want to override the primary method, processRow. But within this method you can reuse a number of other methods, including:

- Using the decodeRow method to process the serialized export data into a data object similar the table rows returned by the VoltDB client interface.
- Iterating over the individual columns in the returned row object to format the data as needed for output

The primary customization in the export decoder class is how the export data is written to its destination. In the case of Hadoop, you will want to use one of the standard Hadoop interfaces to write the formatted data to the Hadoop file system, either as files or as structured data (through Sqoop or similar interfaces).

Whatever output interface is used, you may want to consider buffering the output through a technique similar to the RollingAppender in the Twitter example to optimize disk I/O.

209 Burlington Road, Suite 203
Bedford, MA 01730
Phone: +1.978.528.4660
Fax: +1.978.528.0568
http://voltdb.com

VOLTDB