Design Note for Integrating HCatalog with Hive's ACID Transactions

Eugene Koifman
June 20, 2014

Introduction
MapReduce
    Transaction Manager Requirements
Pig
HCatReader/HCatWriter
Designs Not Chosen

Introduction

Hive 0.13/0.14 introduced the notion of transactions with ACID semantics to Hive. See HIVE-5317 and HIVE-5843 for more information. HCatalog provides a way to read and write Hive data directly from a MapReduce job. This document describes how HCat will be made to participate in Hive transactions. This work is done under HIVE-6207.

MapReduce

HCat supports two basic abstractions to interact with Hive data: HCatInputFormat and HCatOutputFormat. Any MR job which uses HCatInputFormat, by definition, reads Hive data and any MR job using HCatOutputFormat writes Hive data. We must ensure that such jobs make appropriate calls to Hive's Transaction Manager in order to respect/ensure ACID semantics.

Since an MR job may be submitted to the cluster in any number of ways, there isn't guaranteed to be a client that stays around for the duration of the job which could interact with the Transaction Manager. The most logical place to make these calls is in the OutputCommitter of the job. It provides setupJob(), commitJob() and abortJob() lifecycle methods which will be used to begin/end the transactions. It runs from within AppMaster and provides a convenient place from which to send keep-alive messages to the Transaction Manager. When the job is using HCatOutputFormat, we are guaranteed that org.apache.hive.hcatalog.mapreduce.OutputCommitterContainer is used. The later is implemented to make necessary Transaction Manager calls and thus the end user/job author need not do anything special to ensure that the job plays well with ACID.

If a job only uses HCatInputFormat, we cannot guarantee that a specific OutputCommitter is used and thus the user will have to install an OutputCommiter provided by HCat which will make

the appropriate calls to the Transaction Manager and delegate everything else to whichever OutputCommitter the user specifies. If HCatInputFormat-only job does not install the custom OutputCommitter it will run with an equivalent of read-uncommitted isolation level, i.e. with dirty reads. In other words, this job will not acquire any locks itself and will ignore any locks acquired by others, which may result in mid-job failures because, for example, a partition is dropped while being read.

Transaction Manager Requirements

Since the AppMaster may fail and retry the job, we may end up receiving setupJob() event in 1 instance of OutputCommitter and commitJob() in another. Since there is no way to pass any state between these instance, the Transaction Manager API (on the caller side) must be stateless. Transaction Manager should be able to restore/find it's state based on a client ID, which most likely will be the MR job ID.

Pig

Pig script which may be using HCat, at the end of the day ends up generating an MR job which uses HCatInputFormat and HCatOutputFormat and will work the same way. We'll need to modify Pig to make sure that if the job only uses HCatInputFormat, that Pig automatically installs a transaction-aware OutputCommitter.

HCatReader/HCatWriter

Current implementations HCatInputFormatReader/HCatOutputFormatWriter use HCatInputFormat and HCatOutputFormat respectively. The later calls OutputCommitterContainer lifecycle methods. For the former we can just call the Transaction Manager directly. This should in fact be done from HCatReader and have subclasses call super().

Designs Not Chosen

One possibility is to perform transaction lifecycle calls from the client which submits the job instead of the OutputCommitter. MR jobs may be submitted in multiple ways: through Tool, Oozie, WebHCat, something yet to be built. Creating something for each client is difficult to maintain. Also, in case the client dies or in fire-and-forget job submission the job continues to run. So if the Transaction Manager interaction is done from the client, the transaction will simply timeout, but the job will continue to operate on data. Another consideration is that the time between the client submitting a job and the job actually starting to run may be significant for a busy cluster. Thus acquiring the locks on the client may hold locks much longer than necessary affecting throughput.