Links

Motivation

Today, the infrastructure provided by Hive allows for the setup of a single shared warehouse and the authorization model allows for access control within this warehouse if needed. Growth beyond a single warehouse (when datacenter capacity limits are reached) OR separation of capacity usage and allocation requires the creation of multiple warehouses with each warehouse mapping to it's own Hive metastore. Let's define the term physical warehouse to map to a single Hive metastore, the Hadoop cluster it maps to and the data in it.

In organizations with a large number of teams needing a warehouse, there is a need to be able to:

- Maximize sharing of physical clusters to keep operational costs low
- Clearly identify and track capacity usage by teams in the data warehouse

One way to do this is to use a single shared warehouse as we do today, but this has the below issues:

- When the warehouse reaches datacenter capacity limits, it is hard to identify self-contained pieces that can be migrated out.
- Capacity tracking and management becomes an issue.

An alternative is to create a new physical warehouse per team (1:1 mapping), but this is not optimal since the physical resources are not shared across teams, and the operational cost is high. Further, data may not be cleanly partition-able and end up being replicated in multiple physical warehouses.

To provide context, in Facebook, we expect to have 20+ partitions of the warehouse, so operating each in their own physical warehouse will be impractical from an operational perspective.

Requirements

Introduce the notion of a virtual warehouse (namespace) in Hive with the below key properties:

- Can be housed in the same physical warehouse with other virtual warehouses (multi-tenancy).
- Portable (so it can be moved from one physical warehouse to another). Being self-contained is a necessary condition for portability (all queries on this namespace operate only on data available in the namespace).
- Unit of capacity tracking and capacity allocation. This is a nice side effect of creating self-contained namespaces and allows capacity planning based on the virtual warehouse growth.

Mapping many namespaces to 1 physical warehouse keeps the operational cost low. If a physical warehouse reaches capacity limits, portability will allow seamless migration of the namespace to another physical warehouse.

Note that users can operate on multiple namespaces simultaneously although they are likely to most often operate within one namespace. So namespaces are not trying to solve the problem of ensuring that users only have access to a subset of data in the warehouse.

From Hive, therefore the requirements are:

- Provide metadata to identify tables and queries that belong to one namespace.
- Provide controls to prevent operating on tables outside the namespace.
- Provide commands to explicitly request that tables/partitions in namespace1 be made available in namespace2 (since some tables/partitions may be needed across multiple namespaces). Avoid making copies of tables/partitions for this.

Design

The design that is proposed is:

- Modeling namespaces as databases. No explicit accounting/tracking of tables/partitions/views that belong to a namespace is needed since a database provides that already.
- Prevent access using two part name syntax (Y.T). This ensures the database is self-contained.
- Modeling table/partition imports across namespaces using a new concept called Links in Hive. There will be commands to create Links to tables in other databases, alter and drop them. Links do not make copies of the table/partition and hence avoid data duplication in the same physical warehouse.

Let’s take a concrete example:

- Namespace A resides in database A, namespace B in database B.
- Access across these namespace using A.T or B.T syntax is disabled in ‘namespace’ mode.
- The user is importing table T1 from B into A .
- The user issues a CREATE LINK command, which creates metadata in the target namespace A for the table + metadata to indicate which object is linked.
- The ALTER LINK ADD PARTITION command is used to add partitions to the link.
- These partitions are modeled by replicating partition-level metadata in the target database A for the accessible partitions.
- The Link can be dynamic, which means it is kept updated as the source table gets new partitions or drops partitions.

There are 3 alternatives to implementing these ideas in open-source hive and Facebook extensions:

- Implement links as a first-class concept in Hive, and use a Facebook hook to disable Y.T access unless there is a link to the table Y.T.
- Implement links as a first-class concept, and introduce a new syntax T@Y to access linked content. Use a Facebook hook to disable Y.T access.

- Implement links as a first-class concept, and introduce a new syntax T@Y to access linked content. Disable cross database access using a new privilege. All these changes will be in Open Source Hive.

Links to JIRAS for these features:

- HIVE-3047 Add a privilege for cross database access
- HIVE-2989 Adding Table Links to Hive

A basic tenet of our design is that a Hive instance does not operate across physical warehouses. We are building a namespace service external to Hive that has metadata on namespace location across the Hive instances, and allows importing data across Hive instances using replication.

Alternate design options

Modeling Namespace as a Role in Hive (using the authorization model)

The idea here is to create a Role for each namespace and users operating in that namespace belong to that Role. Access to data outside the namespace is made possible by granting permissions to the foreign table/view to the Role for the namespace.

Issues

- A user who belongs to multiple namespaces (and hence multiple roles) will be able to operate on all data across those namespaces at any point in time, so namespaces are no longer self-contained. Imagine the situation of user A who has access to namespaces N1 and N2 running a query on both simultaneously. Either of those queries will be able to access data across both N1 and N2 although this shouldn't be allowed.
- Capacity tracking is more complex
- Operations like show tables, and show partitions do not work without changes.

Modeling Namespace by tagging objects

The idea here is to tag tables/partitions with the namespaces that they belong to. To handle the requirements:

- Tags will have to be explicitly created by a user of the warehouse (through a command)
- Commands like show tables and show partitions will need changes to understand tags.
- Capacity tracking and management will be more complex than using databases for this purpose.
- Data migration is more complex since the data is not contained with the root folder of one database

None of these are insurmountable problems, but using databases to model namespaces is a cleaner approach.
(Taking this idea further, a database in Hive could itself have been implemented using tags in a single global namespace which would have not been as elegant as the current implementation of a database being a first class concept in Hive.)

Modeling Namespace as a database but using views for imports

- The view would be a simple select * using Y.T syntax. It's a degenerate case of view.
- We would need a registry of all views which import tables/partitions from other databases for namespace accounting. This requires adding metadata to these views to distinguish them from other user-created views.
- It would be harder to single instance imports using views (same table/partitions imported twice into the same namespace). Views are too opaque.
  Using partitioned views:
  By definition, there isn't a one-one mapping between a view partition and a table partition. In fact, hive today does not even know about this dependency between view partitions and table partitions. Partitioned views is just a metadata concept - it is not something that the query layer understands. For e.g: if a view V partitioned on ds had 2 partitions: 1 and 2, then a query like select … from V where ds = 3 may still give valid results if the ds=3 is satisfied by the table underlying V. This means that:
- View metadata doesn't stay in sync with source partitions (as partitions get added and dropped). The user has to explicitly do this, which won't work for our case.
- We would like to differentiate the set of partitions that are available for the same imported tables across namespaces. This would require partition pruning based on the view partitions in query rewrite which is not how it works today.

The above notes make it clear that what we are trying to build is a very special case of a degenerate view, and it would be cleaner to introduce a new concept in Hive to model these 'imports'.