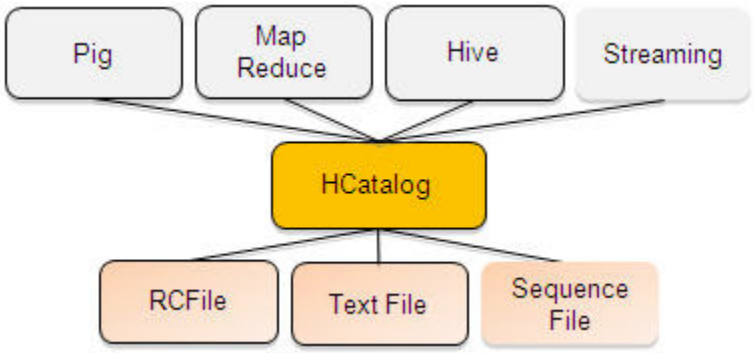
Overview

Table of contents

1 HCatalog

HCatalog is a table and storage management layer for Hadoop that enables users with different data processing tools – Pig, MapReduce, and Hive – to more easily read and write data on the grid. HCatalog's table abstraction presents users with a relational view of data in the Hadoop distributed file system (HDFS) and ensures that users need not worry about where or in what format their data is stored – RCFile format, text files, or sequence files.

HCatalog supports reading and writing files in any format for which a SerDe can be written. By default, HCatalog supports RCFile, CSV, JSON, and sequence file formats. To use a custom format, you must provide the InputFormat, OutputFormat, and SerDe.



2 HCatalog Architecture

HCatalog is built on top of the Hive metastore and incorporates components from the Hive DDL. HCatalog provides read and write interfaces for Pig and MapReduce and uses Hive's command line interface for issuing data definition and metadata exploration commands.

2.1 Interfaces

The HCatalog interface for Pig – HCatLoader and HCatStorer – is an implementation of the Pig load and store interfaces. HCatLoader accepts a table to read data from; you can indicate which partitions to scan by immediately following the load statement with a partition filter statement. HCatStorer accepts a table to write to and optionally a specification of partition keys to create a new partition. You can write to a single partition by specifying the partition key(s) and value(s) in the STORE clause; and you can write to multiple partitions if the partition key(s) are columns in the data being stored. HCatLoader and HCatStorer are implemented on top of HCatInputFormat and HCatOutputFormat, respectively (see HCatalog Load and Store).

The HCatalog interface for MapReduce – HCatInputFormat and HCatOutputFormat – is an implementation of Hadoop InputFormat and OutputFormat. HCatInputFormat accepts a table

to read data from and optionally a selection predicate to indicate which partitions to scan. HCatOutputFormat accepts a table to write to and optionally a specification of partition keys to create a new partition. You can write to a single partition by specifying the partition key(s) and value(s) in the setOutput method; and you can write to multiple partitions if the partition key(s) are columns in the data being stored. (See HCatalog Input and Output.)

Note: There is no Hive-specific interface. Since HCatalog uses Hive's metastore, Hive can read data in HCatalog directly.

Data is defined using HCatalog's command line interface (CLI). The HCatalog CLI supports all Hive DDL that does not require MapReduce to execute, allowing users to create, alter, drop tables, etc. (Unsupported Hive DDL includes import/export, CREATE TABLE AS SELECT, ALTER TABLE options REBUILD and CONCATENATE, and ANALYZE TABLE ... COMPUTE STATISTICS.) The CLI also supports the data exploration part of the Hive command line, such as SHOW TABLES, DESCRIBE TABLE, etc. (see the HCatalog Command Line Interface).

2.2 Data Model

HCatalog presents a relational view of data. Data is stored in tables and these tables can be placed in databases. Tables can also be hash partitioned on one or more keys; that is, for a given value of a key (or set of keys) there will be one partition that contains all rows with that value (or set of values). For example, if a table is partitioned on date and there are three days of data in the table, there will be three partitions in the table. New partitions can be added to a table, and partitions can be dropped from a table. Partitioned tables have no partitions at create time. Unpartitioned tables effectively have one default partition that must be created at table creation time. There is no guaranteed read consistency when a partition is dropped.

Partitions contain records. Once a partition is created records cannot be added to it, removed from it, or updated in it. Partitions are multi-dimensional and not hierarchical. Records are divided into columns. Columns have a name and a datatype. HCatalog supports the same datatypes as Hive (see HCatalog Load and Store).

3 Data Flow Example

This simple data flow example shows how HCatalog can help grid users share and access data.

First Joe in data acquisition uses distcp to get data onto the grid.

```
hadoop distcp file:///file.dat hdfs://data/rawevents/20100819/data
```

```
hcat "alter table rawevents add partition (ds='20100819') location 'hdfs://data/
rawevents/20100819/data'"
```

Second Sally in data processing uses Pig to cleanse and prepare the data.

Without HCatalog, Sally must be manually informed by Joe when data is available, or poll on HDFS.

```
A = load '/data/rawevents/20100819/data' as (alpha:int, beta:chararray, …);
B = filter A by bot_finder(zeta) = 0;
…
store Z into 'data/processedevents/20100819/data';
```

With HCatalog, HCatalog will send a JMS message that data is available. The Pig job can then be started.

```
A = load 'rawevents' using HCatLoader();
B = filter A by date = '20100819' and by bot_finder(zeta) = 0;
…
store Z into 'processedevents' using HCatStorer("date=20100819");
```

Third Robert in client management uses Hive to analyze his clients' results.

Without HCatalog, Robert must alter the table to add the required partition.

```
alter table processedevents add partition 20100819 hdfs://data/processedevents/20100819/
data

select advertiser_id, count(clicks)
from processedevents
where date = '20100819'
group by advertiser_id;
```

With HCatalog, Robert does not need to modify the table structure.

```
select advertiser_id, count(clicks)
from processedevents
where date = ‘20100819’
group by advertiser_id;
```