HORTONWORKS®

Transactional Operations in Apache Hive

- Eugene Koifman

DataWorks Summit, San Jose 2018

Agenda

- A bit of history
- Current Functionality
- Design
- Future Plans
- Closing Remarks

Early Hive

- Transactions
  - ACID: Atomicity, Consistency, Isolation, Durability
- Atomicity - Rely on File System ‘rename’
  - Insert into T partition(p=1) select …. - OK
  - Dynamic Partition Write – not OK
  - Multi-Insert statement – not OK
    - FROM <expr> Insert into A select … Insert Into B select …
- Isolation - Lock Manager
  - S/X locks – not good for long running analytics

Early Hive – Changing Existing Data

- Drop <…>
- Insert Overwrite = Truncate + Insert
  - Gets expensive if done often on small % of data

Goals

- Support ACID properties
- Support SQL Update/Delete/Merge
- Low rate of transactions
  - Not OLTP
  - Not a replacement for MySql or HBase

Features – Hive 3

Transactional Tables

- Not all tables support transactional semantics
  - Managed Tables
  - No External tables or Storage Handler (Hbase, Druid, etc)
- Fully ACID compliant
- Single statement transactions
- Cross partition/cross table transactions
- Snapshot Isolation
  - Between Serializable and Repeatable Read

Transactional Tables – Full CRUD

- Supports Update/Delete/Merge
- CREATE TABLE T(a int, b int) STORED AS ORC TBLPROPERTIES ('transactional'='true');
- Restrictions
  - Managed Table
  - Table cannot be sorted
  - Currently requires ORC File but anything implementing
    - AcidInputFormat/AcidOutputFormat
  - Bucketing is optional!
- If upgrading from Hive 2
  - Requires Major Compaction before Upgrading

Transactional Tables – Insert only

- CREATE TABLE T(a int, b int) TBLPROPERTIES ('transactional'='true', ‘transactional_properties’=‘insert_only’);
  - Managed Table
  - Any storage format

Transactional Tables – Convert from flat tables

- ALTER TABLE T SET TBLPROPERTIES ('transactional'='true')
- ALTER TABLE T(a int, b int) SET TBLPROPERTIES ('transactional'='true’, ‘transactional_properties’=‘true’);
  - Metadata Only operation
  - Compaction will eventually rewrite the table

Transactional Tables - New In Hive 3

- Alter Table Add Partition…
- Alter Table T Concatenate
- Alter Table T Rename To….
- Export/Import Table
- Non-bucketed tables
- Load Data… Into Table …
- Insert Overwrite
- Fully Vectorized
- Create Table As …
- LLAP Cache
- Predicate Push Down

Design – Hive 3

Transactional Tables – Insert Only

- Transaction Manager
  - Begin transaction and obtain a Transaction ID
  - For each table, get a Write ID – determines location to write to

```
create table TM (a int, b int) TBLPROPERTIES
  ('transactional'='true',
    'transactional_properties'='insert_only');

insert into TM values(1,1);
~~insert into TM values(2,2);~~
insert into TM values(3,3);
```

```
tm
 ── delta_0000001_0000001_0000
   └── 000000_0
 ── delta_0000002_0000002_0000
   └── 000000_0
 ── delta_0000003_0000003_0000
    └── 000000_0
```

Transaction Manager

- Transaction State
  - Open, Committed, Aborted
- Reader at Snapshot Isolation
  - A snapshot is the state of all transactions
  - High Water Mark + List of Exceptions

```
tm
 ── delta_0000001_0000001_0000
   └── 000000_0
 ~~── delta_0000002_0000002_0000~~
   ~~└── 000000_0~~
 ── delta_0000003_0000003_0000
    └── 000000_0
```

- Atomicity & Isolation

Full CRUD

- No in-place Delete - Append-only file system
- Isolate readers from writers

ROW__ID

- CREATE TABLE acidtbl (a INT, b STRING) STORED AS ORC TBLPROPERTIES ('transactional'='true');

Metadata Columns	**original_write_id** **bucket_id** **row_id** current_write_id **ROW__ID**
User Columns	col_1: a : INT col_2: b : STRING

Create

- INSERT INTO acidtbl (a,b) VALUES (100, “foo”), (200, “xyz”), (300, “bee”);

ROW__ID	a	b
{ 1, 0, 0 }	100	“foo”
{ 1, 0, 1 }	200	“xyz”
{ 1, 0, 2 }	300	“bee”

delta_00001_00001/bucket_0000

Delete

- DELETE FROM acidTbl where a = 200;

ROW__ID	a	b
{ 1, 0, 0 }	100	“foo”
{ 1, 0, 1 }	200	“xyz”
{ 1, 0, 2 }	300	“bee”

delta_00001_00001/bucket_0000

ROW__ID	a	b
{ 1, 0, 1 }	null	null

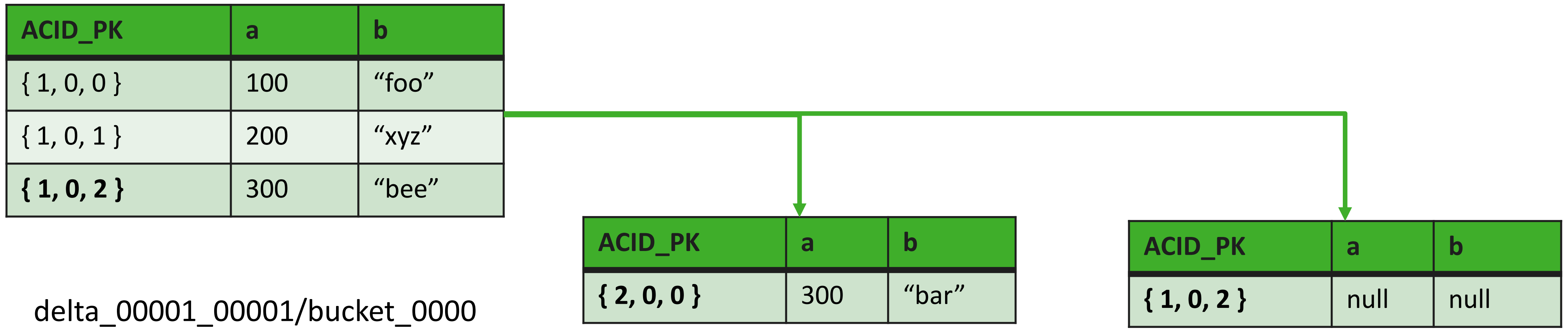
delete_delta_00002_00002/bucket_0000

- Readers skip deleted rows

HORTONWORKS

Update

- Update = delete + insert
- UPDATE acidTbl SET b = “bar” where a = 300;



ACID_PK	**a**	**b**
{ 1, 0, 0 }	100	“foo”
{ 1, 0, 1 }	200	“xyz”
{ 1, 0, 2 }	300	“bee”

delta_00001_00001/bucket_0000

ACID_PK	**a**	**b**
{ 2, 0, 0 }	300	“bar”

delta_00003_00003/bucket_0000

ACID_PK	**a**	**b**
{ 1, 0, 2 }	null	null

delete_delta_00003_00003/bucket_0000

Read

- Ask Transaction Manager for Snapshot Information
  - Decide which deltas are relevant
- Take all the files in delta_x_x/ and split them into chunks for each processing Task to work with
- Localize all delete events from each delete_deleta_x_x/ to each task
  - Highly Compressed with ORC
- Filter out all Insert events that have matching delete events
  - Requires an Acid aware reader – thus AcidInputFormat

Design - Compactor

- More Update operations = more delete events – make reads more expensive
- Insert operations don’t add read overhead

Design - Compactor

- Compactor rewrites the table in the background
  - Minor compaction - merges delta files into fewer deltas
  - Major compactor merges deltas with base - more expensive
  - This amortizes the cost of updates and self tunes the tables
    - Makes ORC more efficient - larger stripes, better compression
- Compaction can be triggered automatically or on demand
  - There are various configuration options to control when the process kicks in.
  - Compaction itself is a Map-Reduce job
- Key design principle is that compactor does not affect readers/writers
- Cleaner process – removes obsolete files
- Requires Standalone metastore

Merge Statement – SQL Standard 2011 (Hive 2.2)

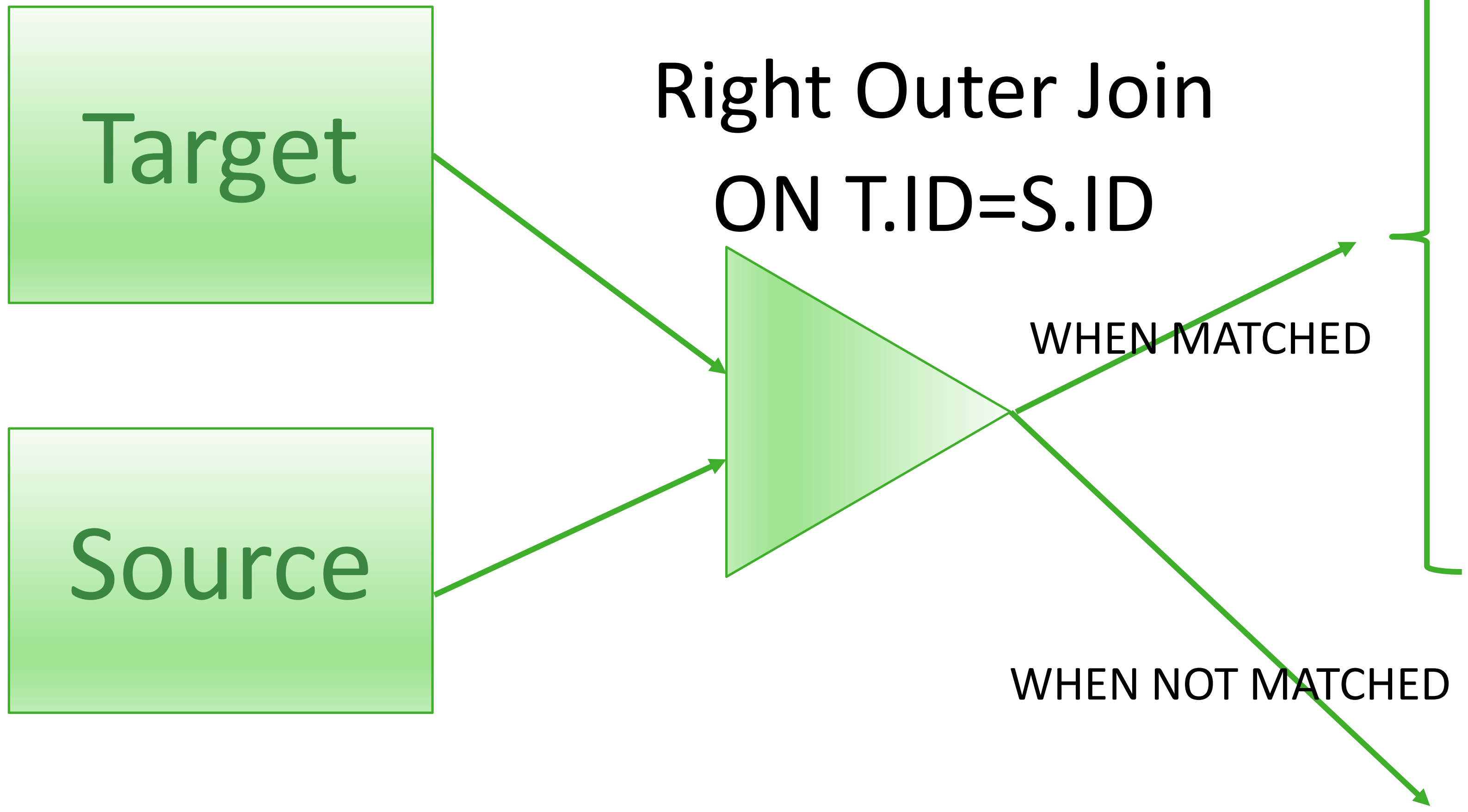
ID	State	County	Value
1	CA	LA	19.0
2	MA	Norfolk	15.0
7	MA	Suffolk	50.15
16	CA	Orange	9.1

```
MERGE INTO  TARGET T
            USING SOURCE S ON T.ID=S.ID
                 WHEN MATCHED THEN
                 UPDATE SET T.Value=S.Value
                 WHEN NOT MATCHED
                     INSERT (ID,State,Value)
                 VALUES(S.ID, S.State, S.Value)
```

ID	State	Value
1		20.0
7		80.0
100	NH	6.0

ID	State	County	Value
1	CA	LA	20.0
2	MA	Norfolk	15.0
7	MA	Suffolk	80.0
16	CA	Orange	9.1
100	NH	null	6.0

SQL Merge



ACID_PK	ID	Stat e	County	Value
{ 2, 0, 1 }	1	CA	LA	20.0
{ 2, 0, 2 }	7	MA	Suffolk	80.0

delta_00002_00002/bucket_0000

ACID_PK	Data
{ 1, 0, 1 }	null
{ 1, 0, 3 }	null

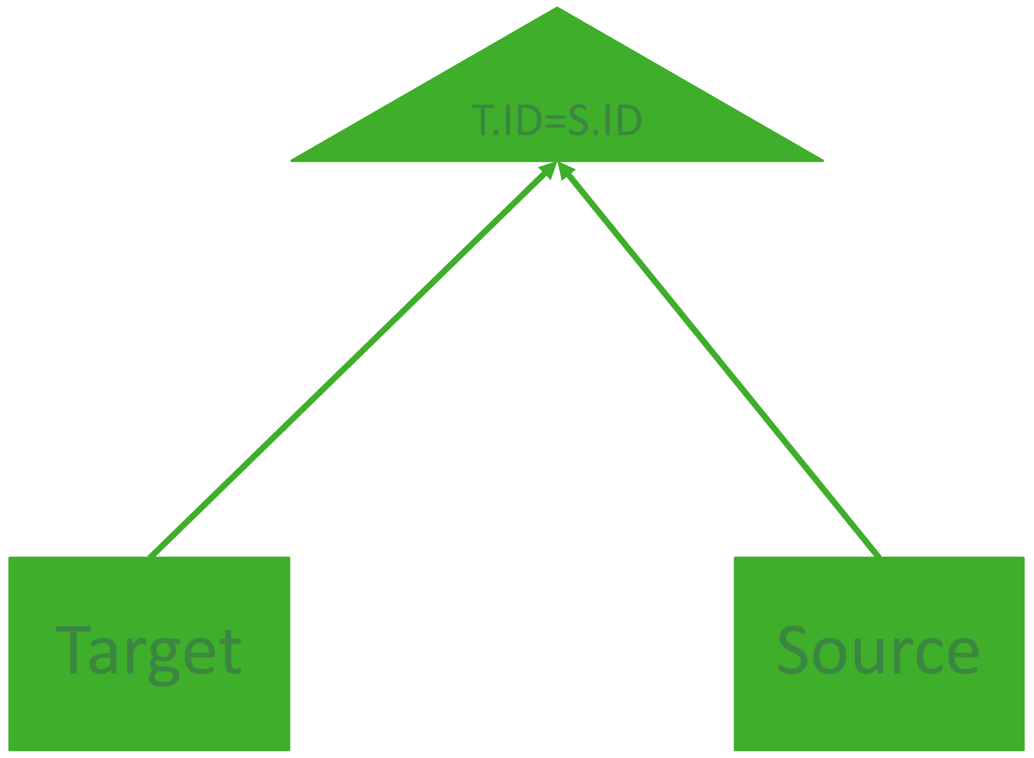
delete_delta_00002_00002/bucket_0000

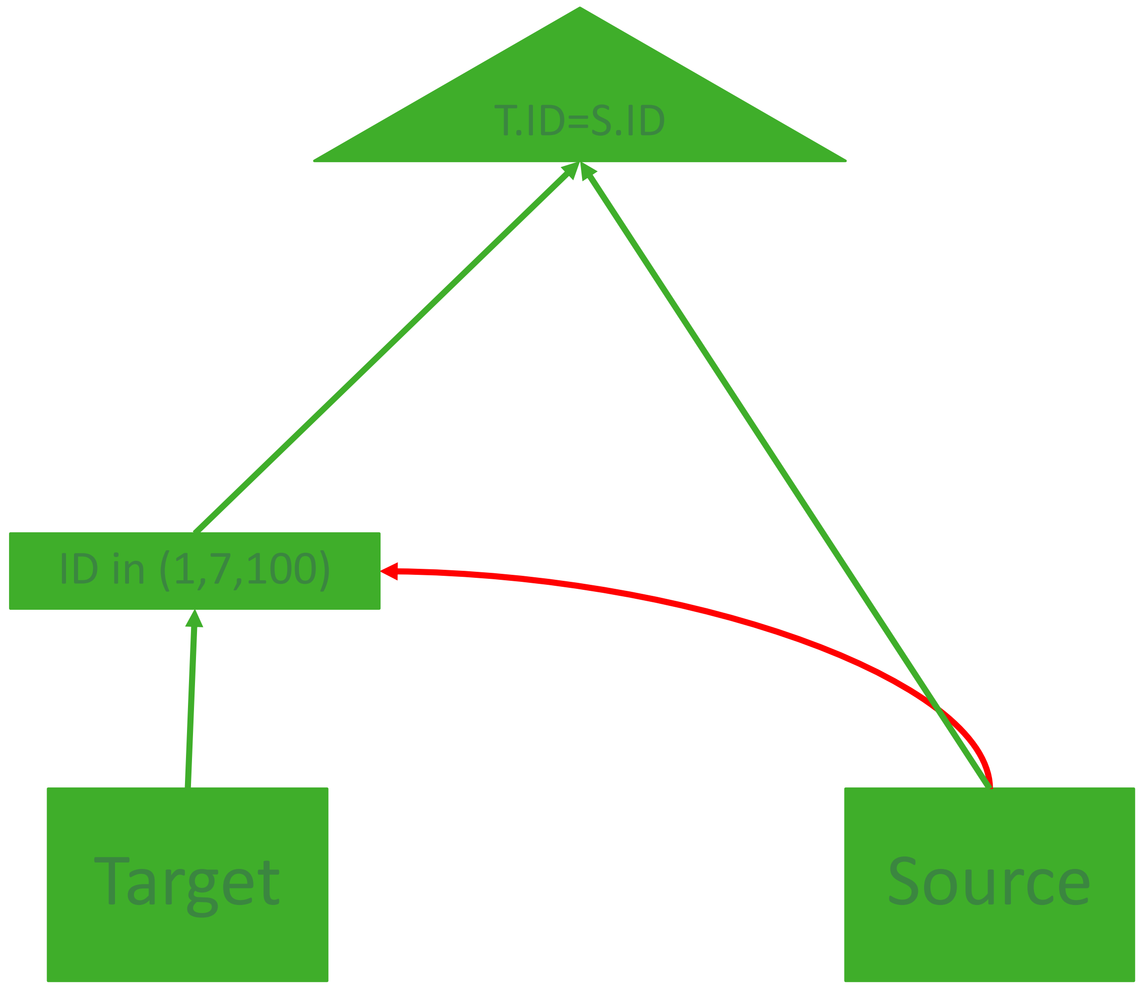
ACID_PK	ID	State	County	Value
{ 2, 0, 1 }	100	NH		6.0

delta_00002_00002_001/bucket_0000

Merge Statement Optimizations

- Semi Join Reduction
  - aka Dynamic Runtime Filtering
  - On Tez only





Design - Concurrency

- Inserts are never in conflict since Hive does not enforce unique constraints
- Write Set tracking to prevent Write-Write conflicts in concurrent transactions
- Lock Manager
  - DDL operations acquire eXclusive locks – metadata operations
  - Read operations acquire Shared locks

Tooling

- SHOW COMPACTIONS
  - Hadoop Job ID
- SHOW TRANSACTIONS
- SHOW LOCKS
  - What a lock is blocked on
- ABORT TRANSACTIONS txnid1, txnid2....

Other Subsystems

- Result Set Caching
  - Is it valid for current reader?
- Materialized Views
  - Incremental View Manitenance
- Spark
  - HiveWarehouseConnector: HS2 + LLAP

Streaming Ingest API

- Connection – Hive Table
  - Begin transaction
  - Commit/Abort transaction
  - org.apache.hive.streaming.StreamingConnection
- Writer
  - Write records
  - org.apache.hive.streaming.RecordWriter
- Append Only via this API
  - Update/Delete via SQL
- Optimized for Write operations
- Requires more aggressive Compaction for efficient reads
- Supports dynamic partitioning in a single transaction

Limitations

- Transaction Manager
  - State is persisted in the metastore RDBMS
- Begin/Commit/Abort
  - Metastore calls

Future

HORTONWORKS

Future Work

- Multi statement transactions, i.e. BEGIN TRANSACTION/COMMIT/ROLLBACK
- Performance
  - Smarter Compaction
- Finer grained concurrency management/conflict detection
- Read Committed w/Lock Based scheduling
- Better Monitoring/Alerting
- User define Primary Key
  - Transactional Tables sorted on PK

Further Reading

Etc

- Documentation
  - https://cwiki.apache.org/confluence/display/Hive/Hive+Transactions
  - https://cwiki.apache.org/confluence/display/Hive/Streaming+Data+Ingest+V2
- Follow/Contribute
  - https://issues.apache.org/jira/browse/HIVE-14004?jql=project%20%3D%20HIVE%20AND%20component%20%3D%20Transactions
- user@hive.apache.org
- dev@hive.apache.org

Credits

- Alan Gates
- Sankar Hariappan
- Prasanth Jayachandran
- Eugene Koifman
- Owen O’Malley
- Saket Saurabh
- Sergey Shelukhin
- Gopal Vijayaraghavan
- Wei Zheng

Thank You

HORTONWORKS