Unpacking the Transaction Log

Burak Yavuz and Denny Lee

Who are we?

databricks

- Software Engineer – Databricks

  “We make your streams come true”

- Apache Spark™ Committer
- MS in Management Science & Engineering - Stanford University
- BS in Mechanical Engineering - Bogazici University, Istanbul

Who are we?

- Developer Advocate – Databricks
- Working with Apache Spark™ since v0.6
- Former Senior Director Data Science Engineering at Concur
- Former Microsoftie: Cosmos DB, HDInsight (Isotope)
- Masters Biomedical Informatics - OHSU
- BS in Physiology - McGill

Outline

- The Delta Log (Transaction Log)
  - Contents of a commit
  - Optimistic Concurrency Control
  - Computing / updating the state of a Delta Table
- Time Travel
- Demo

Delta On Disk

Transaction Log

Table Versions

(Optional) Partition Directories

Data Files

Table = result of a set of actions

Update Metadata – name, schema, partitioning, etc

Add File – adds a file (with optional statistics)

Remove File – removes a file

Set Transaction – records an idempotent txn id

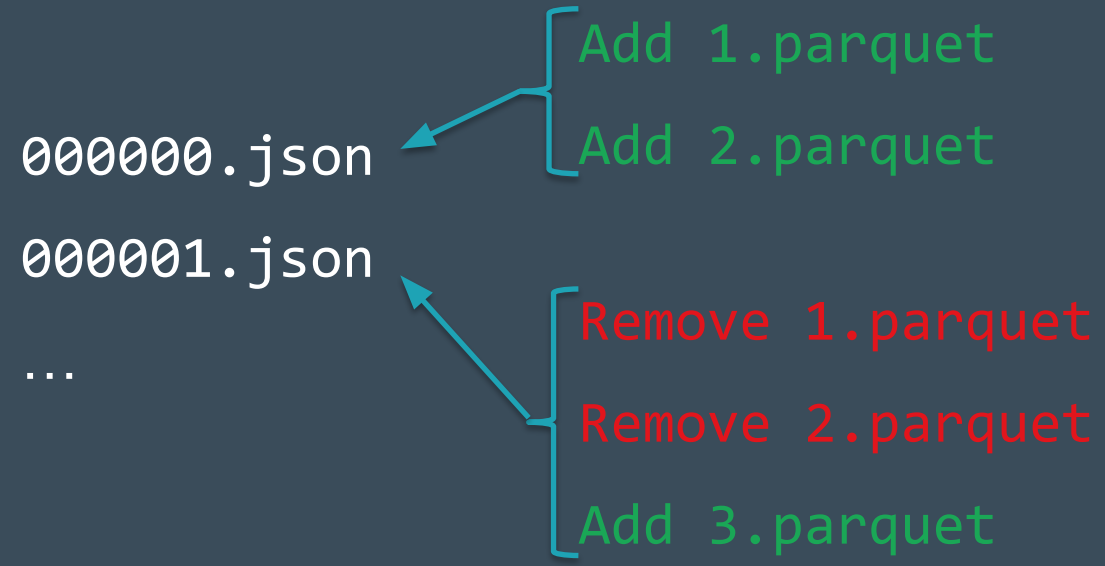
Change Protocol – upgrades the version of the txn protocol

Commit Info – information around commit for auditing

Result: Current Metadata, List of Files, List of Txns, Version

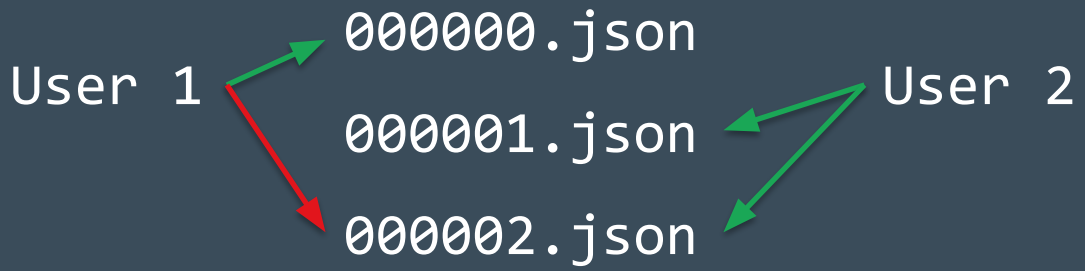
Implementing Atomicity

Changes to the table are stored as ordered, atomic units called commits



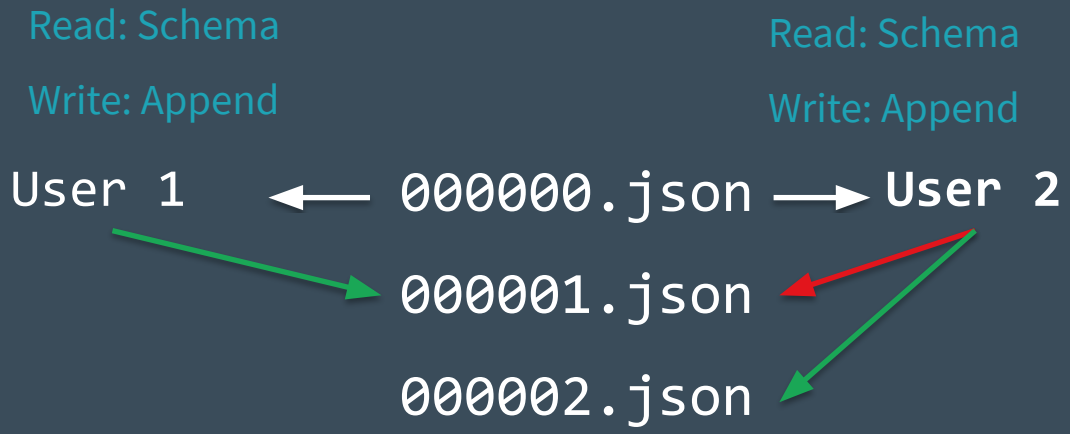
Ensuring Serializablity

Need to agree on the order of changes, even when there are multiple writers.



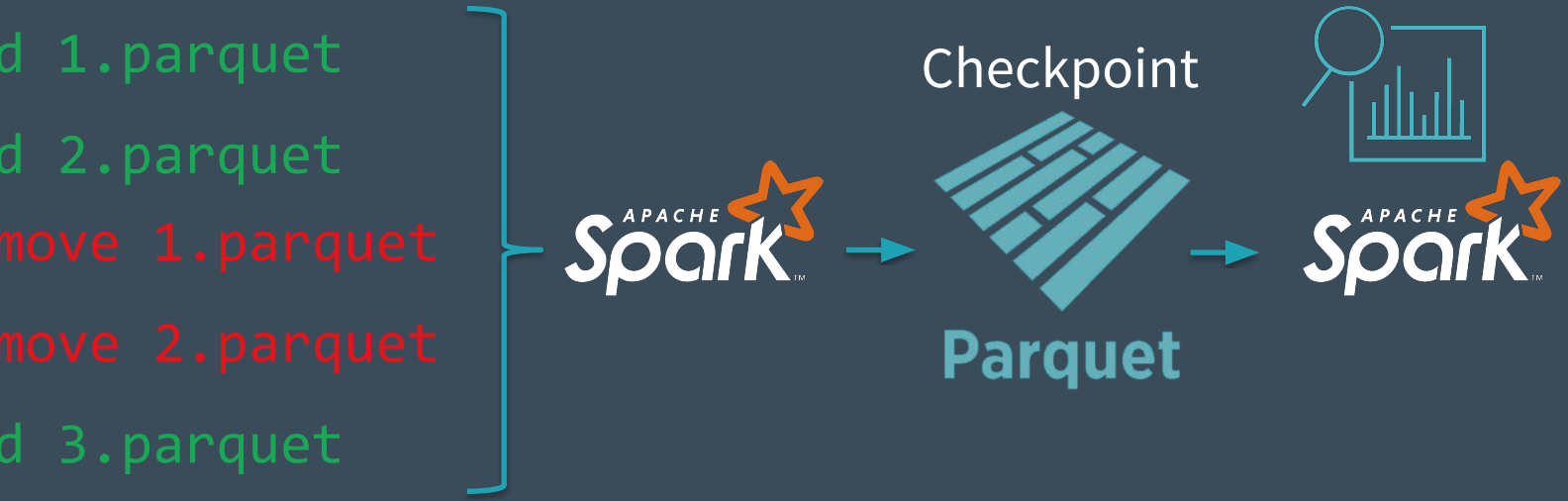
Solving Conflicts Optimistically

1. Record start version
2. Record reads/writes
3. Attempt commit
4. If someone else wins, check if anything you read has changed.
5. Try again.



Handling Massive Metadata

Large tables can have millions of files in them! How do we scale the metadata? Use Spark for scaling!

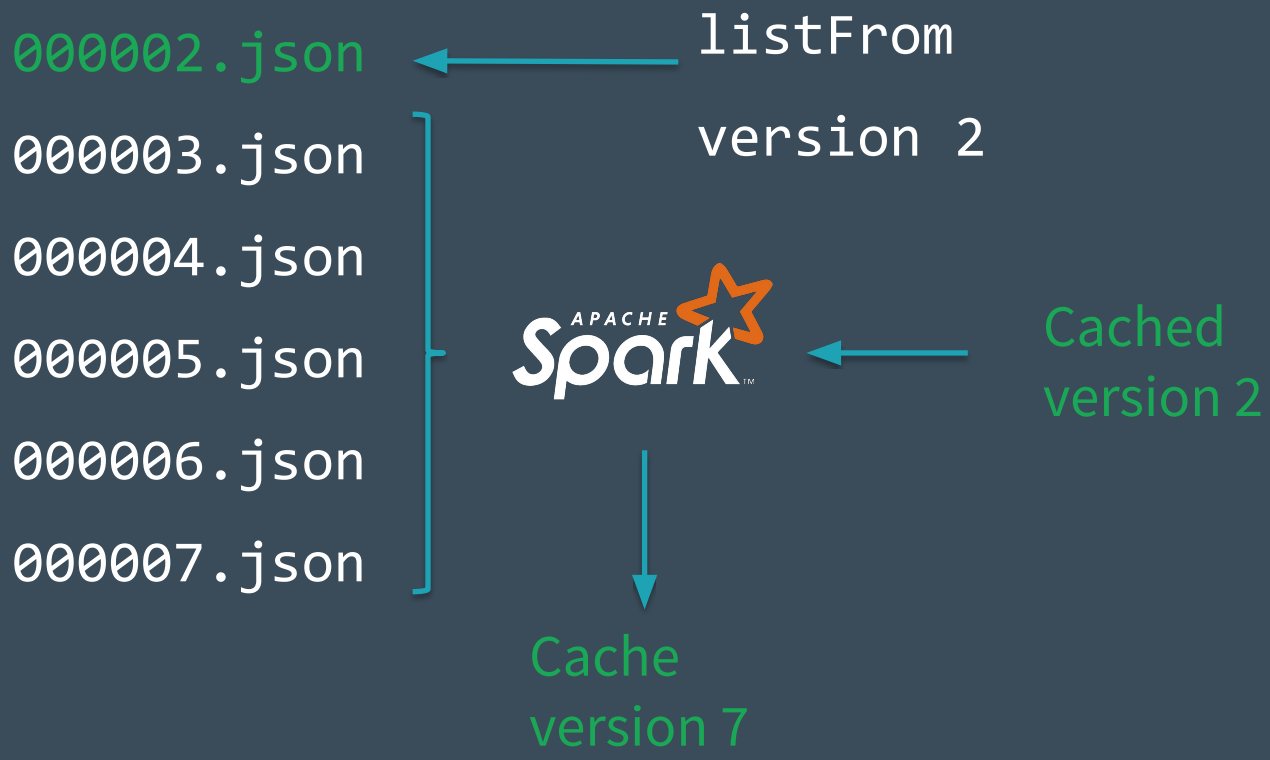


Computing Delta's State

000000.json
000001.json
000002.json

Cache
version 2

Updating Delta's State



Updating Delta's State

000007.json ← listFrom version 7

000008.json

000009.json

0000010.json

0000010.checkpoint.parquet

0000011.json

0000012.json

→ Cache version 12

Outline

- The Delta Log (Transaction Log)
- Time Travel
  - How it works
  - Limitations
- Demo

Time Travelling by version

```
SELECT * FROM my_table VERSION AS OF 1071;
SELECT * FROM my_table@v1071  -- no backticks to specify @
spark.read.option("versionAsOf", 1071).load("/some/path")
spark.read.load("/some/path@v1071")
```

```
deltaLog.getSnapshotAt(1071)
```

Time Travelling by timestamp

```
SELECT * FROM my_table TIMESTAMP AS OF '1492-10-28';
SELECT * FROM my_table@1492102800000000 -- yyyyMMddHHmmssSSS
spark.read.option("timestampAsOf", "1492-10-28").load("/some/path")
spark.read.load("/some/path@1492102800000000")
```

```
deltaLog.getSnapshotAt(1071)
```

Time Travelling by timestamp

Commit timestamps come from storage system modification timestamps

```
001070.json     375-01-01
001071.json    1453-05-29
001072.json    1923-10-29
001073.json    1920-04-23
```

Time Travelling by timestamp

Timestamps can be out of order. We adjust by adding 1 millisecond to the previous commit's timestamp.

```
001070.json     375-01-01   375-01-01
001071.json    1453-05-29  1453-05-29
001072.json    1923-10-29  1923-10-29
001073.json    1920-04-23  1923-10-29 00:00:00.001
```

Time Travelling by timestamp

Price is right rules: Pick closest commit with timestamp that doesn't exceed the user's timestamp.

```
001070.json    375-01-01
001071.json   1453-05-29   <-- 1492-10-28
001072.json   1923-10-29
001073.json   1923-10-29 00:00:00.001


deltaLog.getSnapshotAt(1071)
```

Time Travel Limitations

- Requires transaction log files to exist
  - delta.logRetentionDuration = "interval <interval>”
- Requires data files to exist
  - delta.deletedFileRetentionDuration = "interval <interval>"
  - If you Vacuum, you lose data
- Therefore time travel in order of months/years infeasible
  - Expensive storage
  - Computing Delta’s state won’t scale

Demo

Thank You

“Do you have any questions for my prepared answers?”

– Henry Kissinger