Checkpoint & Recovery

Mengyang Lyu, Yuning Zhang, Zhaozhe Song

Goals

75% goal

- Consistent complete checkpoint of tables
- Checkpoint of catalogs (partial)
- Synchronous scanning and writing to file

100% goal

- Recovery from checkpoint and logs
- Async scanning and writing during checkpointing

125% goal

- Explore performance optimizations (multi-threading, logger / checkpointer thread allocation, reverse recovery order)

Project Goals Re-iteration

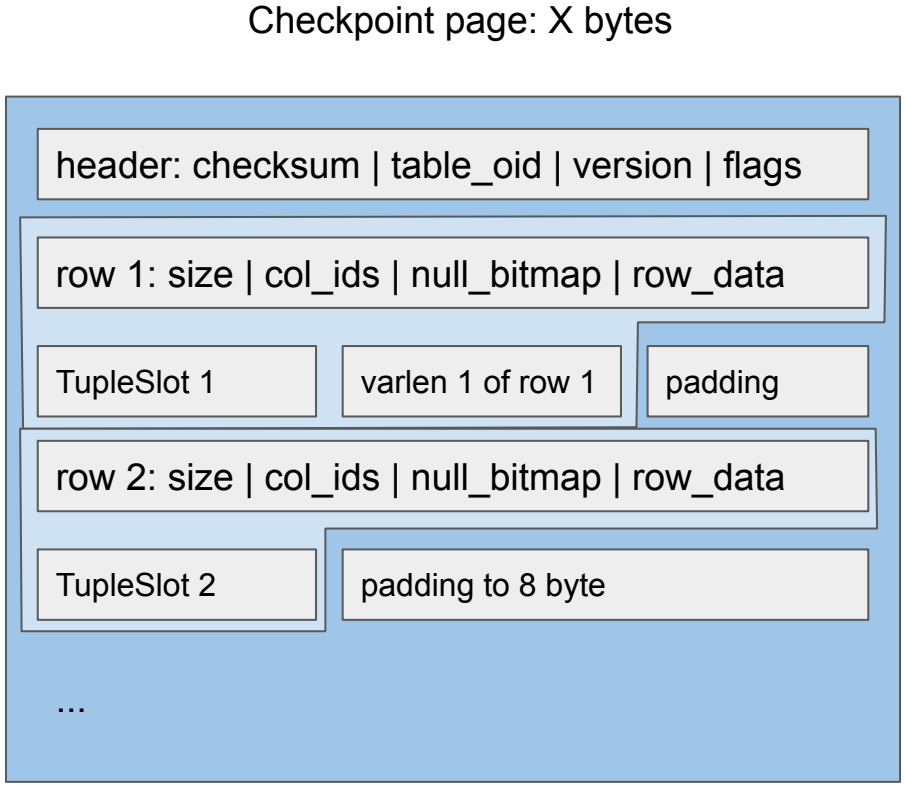
Consistent complete checkpoint of catalogs and tables(75%)

- User tables: Finished
- Catalog: Able to recover catalog tables, but not catalog
  - catalog tables are automatically created and populated upon creation
  - we need APIs to
    - create catalog tables automatically without populate it
    - a single API to update internal data structures in catalog and create user tables.

Project Goals Re-iteration

Synchronous scanning and writing to file (75 %)

- version field in header
  - schema version for user table, or,
  - catalog table type.
- flags field in header
  - currently only IS_CATALOG is used
- TupleSlot field in row data
  - Used for log recovery
- Huge rows are also supported!
  - implemented in reader/writer
  - transparent to upper level



Project Goals Re-iteration

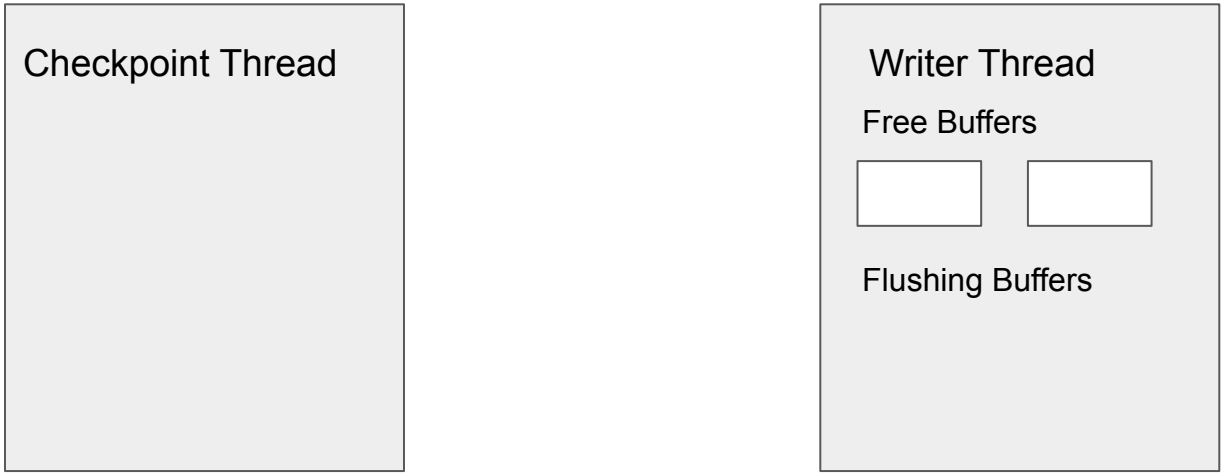
Recovery from checkpoint and logs (100 %)

- Recovery from checkpoints
- Modified and merged legacy logging PR to handle varlens
- Recovery from logs

- We support only single table checkpointing and recovery
  - Catalog support is not finished
  - Table oids in logging are hard-coded in the system

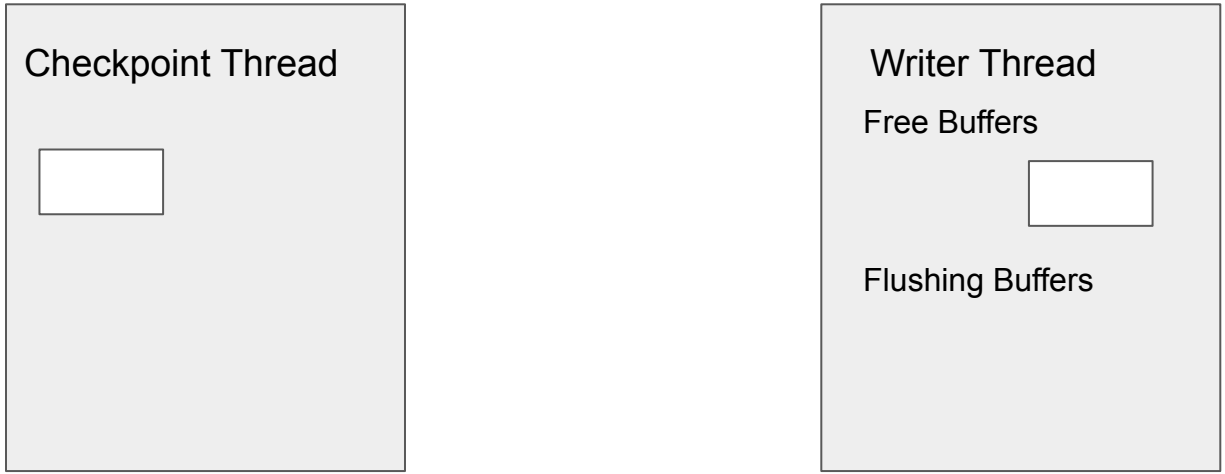
Project Goals Re-iteration

- Async scanning and writing during checkpointing



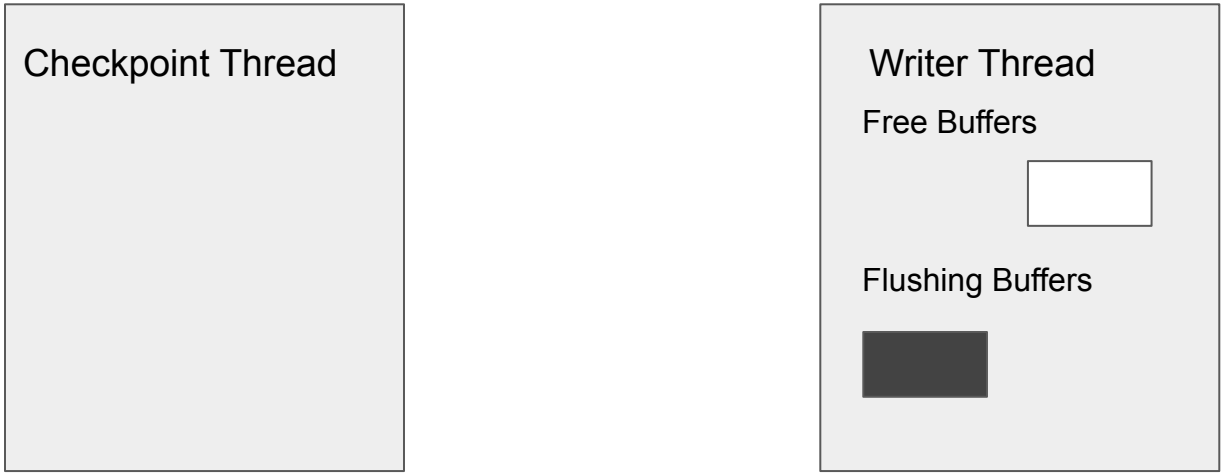
Project Goals Re-iteration

- Async scanning and writing during checkpointing



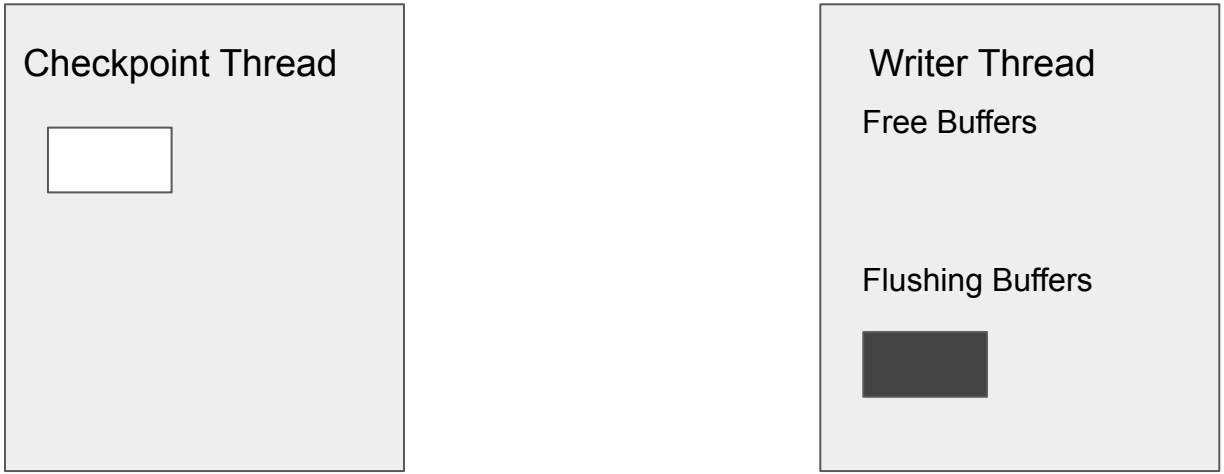
Project Goals Re-iteration

- Async scanning and writing during checkpointing



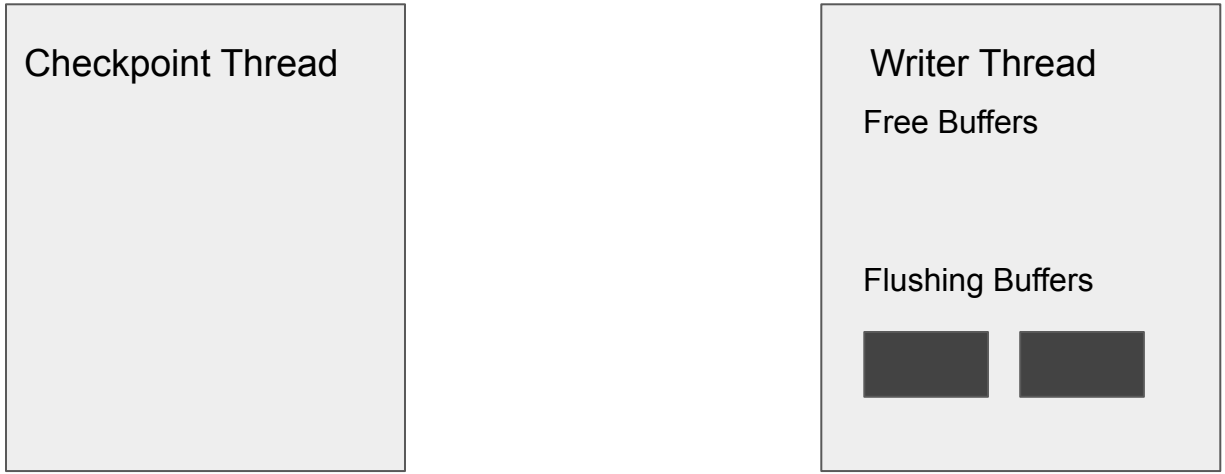
Project Goals Re-iteration

- Async scanning and writing during checkpointing



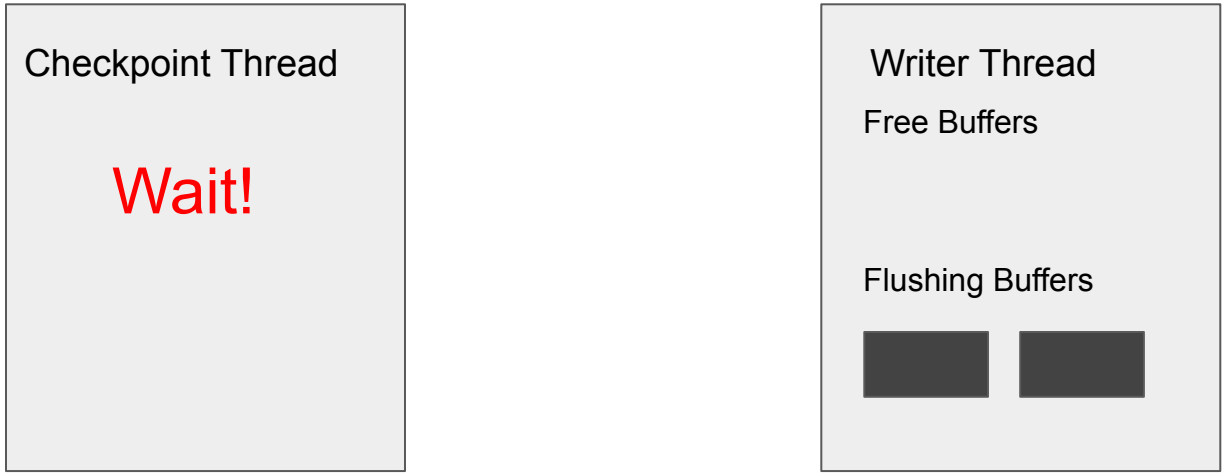
Project Goals Re-iteration

- Async scanning and writing during checkpointing



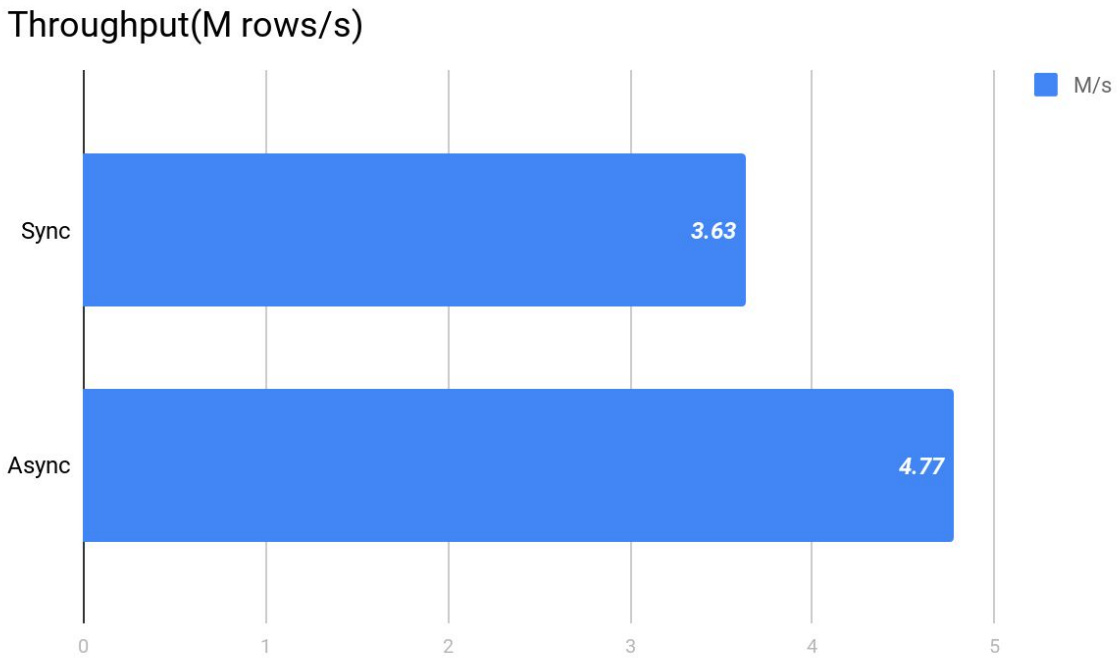
Project Goals Re-iteration

- Async scanning and writing during checkpointing



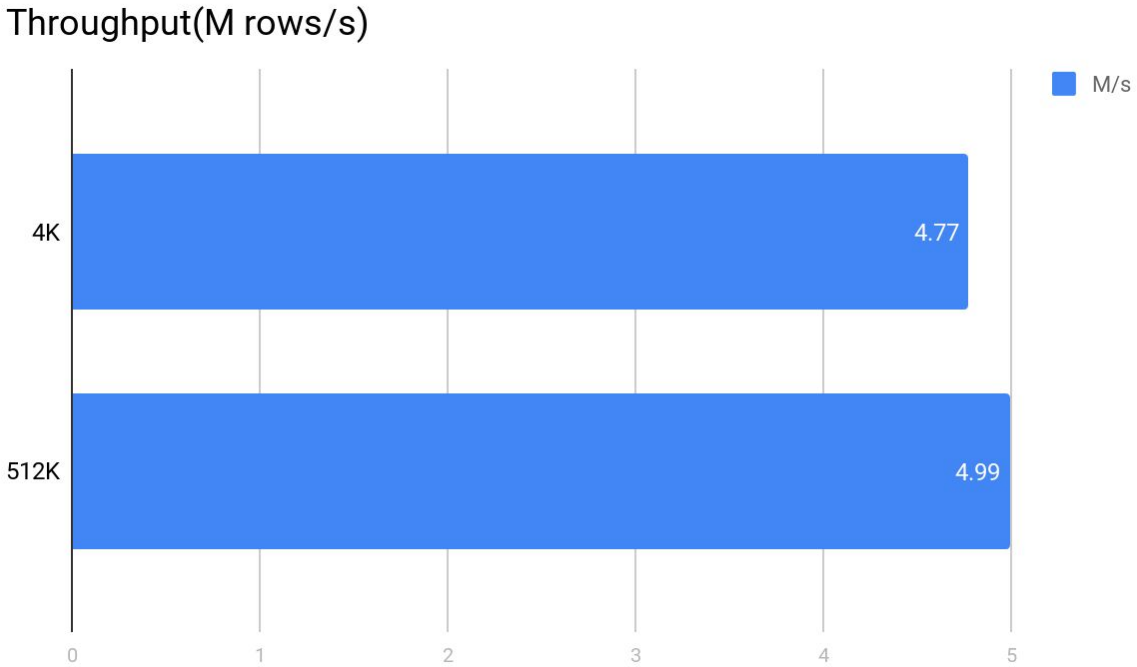
Benchmark results

From Sync to Async



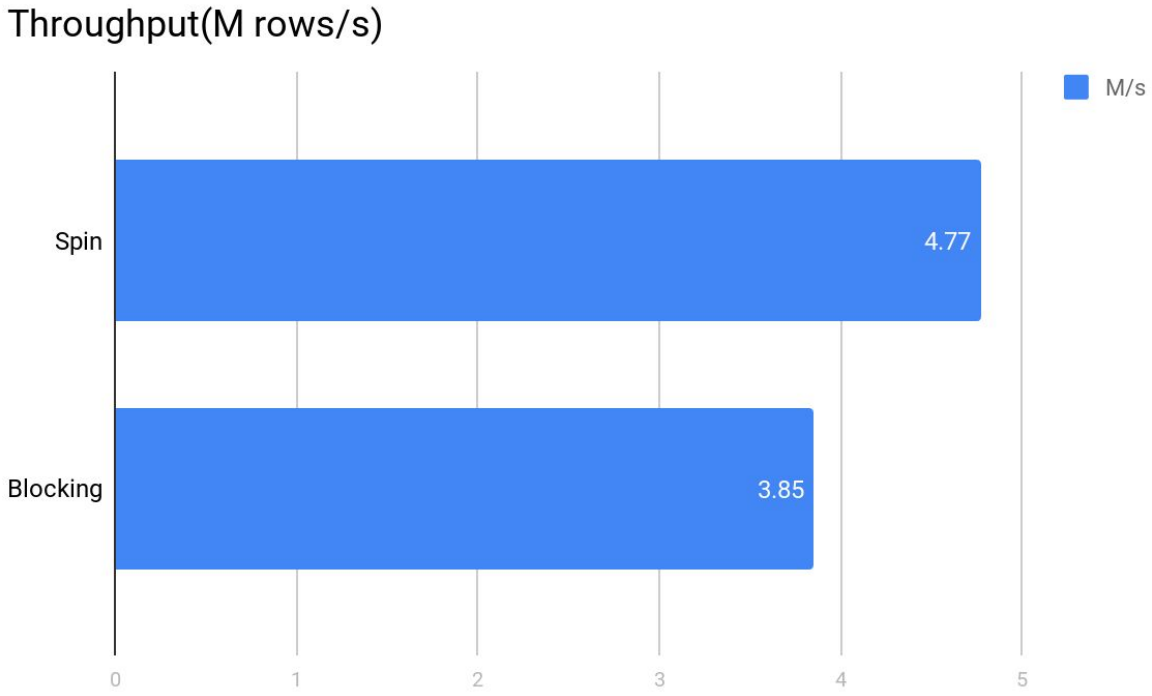
Benchmark results

Altering buffer size



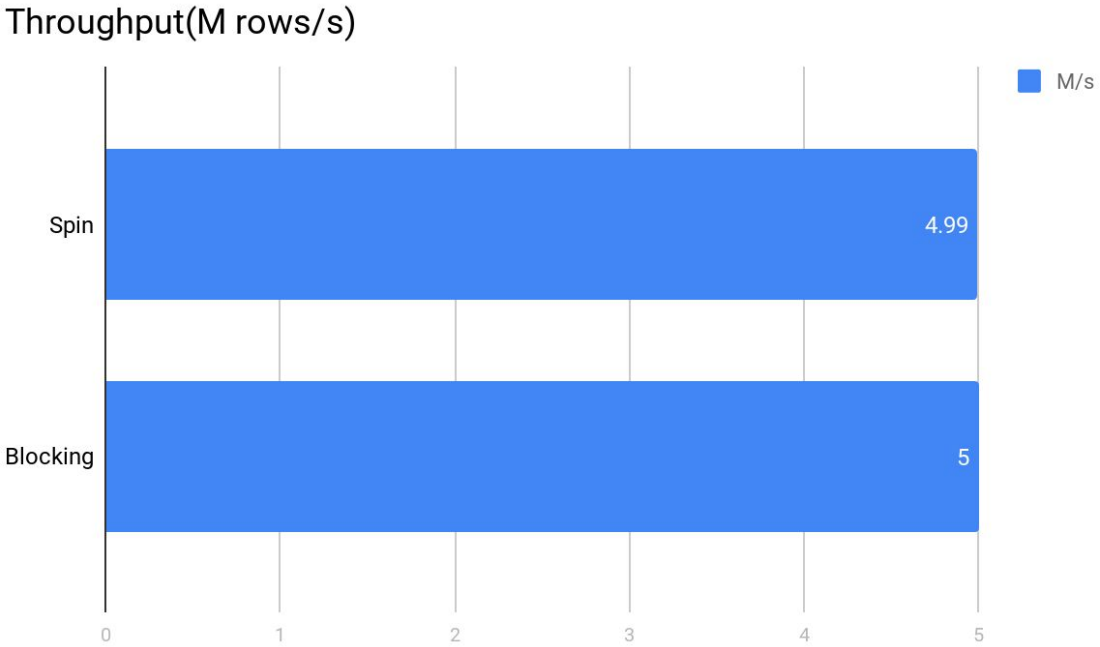
Benchmark results

Spin or wait?



Benchmark results

Spin or wait? (512K)



Tests

Basic idea - Scan the table before and after recovery

Checkpoint & recovery

- Single table checkpoint & recovery with/without varlens.
- Full database recovery for catalogs and all tables [WIP](need catalog)

Log & recovery

- Logging tests
- Log recovery from empty table
- Log recovery from checkpoint state

Code Quality

- Checkpointing & recovery & writer/reader code **good**
  - rather independent from other components
  - well documented
- Logging and recovery **weak**
  - Future migration to SqlTables
  - Transactions store a hard-coded table oid
- Interaction with catalog **weak**
  - Have to follow up with catalog implementation

Existing problems

- Logging requires re-write
  - Use SqlTable layer API instead of DataTable
    - DataTable* -> SqlTable*
    - Layout* -> Schema*
  - Same problem for the larger transaction/test infrastructure
    - We did not change it yet because it will break many tests
    - We used a hack to do testing
- Interaction with catalog [WIP]
  - To recover catalog
    - system starts up
    - system catalog tables created, but not populated
    - recovery will read the logs and populate the catalog tables.
  - To use catalog information to recover all tables automatically

Future works

- Comprehensive migration of all transaction/testing infra to SqlTable
  - Affect logging and recovery
- Interaction with catalog
- Use keys in logging/checkpoints instead of TupleSlot

Performance Optimization

- Multi-thread checkpoint: partition a table using range information from index
- logger/checkpointer thread allocation: read # of threads from settings manager, and start/stop logger/checkpointer threads automatically.