Towards Accurate and Fast Evaluation of Multi-Stage Log-Structured Designs
Hyeontaek Lim (CMU), David G. Andersen (CMU), Michael Kaminsky (Intel Labs)

Multi-Stage Log-Structured Design Evaluation
- Multiple stages of append-only logs to segregate fresh and old data
- Many system designs:
  - LevelDB, RocksDB, BigTable, HBase, Cassandra, ...
- Developers need tools for accurate and fast evaluation
  - Which design is best for this workload?
  - How should the systems’ parameters be set?
  - How sensitive is that choice to changes in workloads?

Problems of Prior Evaluation Methods
- Asymptotic analysis: **Not very accurate**
  - E.g., O(log N) of insert cost often overestimates real cost
- Experiment: **Slow and often hard to generalize**
  - E.g., Obtaining “12 k inserts/sec” may take hours to days

Solution: New Analytic Primitives
- Unique, Unique⁻¹, Merge
- Convert between # of requests and # of unique keys
- Consider redundancy in the workload for high accuracy
- Allow building system models (not shown) to estimate performance metrics
  - How often do table merges occur?
  - How much data do they write?

Example Design: LevelDB
- LevelDB-ana: Our LevelDB model
  - 0.01 sec/eval for 100 M unique keys (orders of magnitude faster)
- LevelDB-sim: Our lightweight C++ LevelDB simulator
  - 12 mins/eval for 100 M unique keys
- LevelDB-impl: Full LevelDB implementation
  - 2.9 hours/eval for 10 M unique keys

Details of New Analytic Primitives
- Let $K$ be a set of unique keys, $f_X(k)$ be key $k$’s probability in each insert request
  - $\text{Unique}(p)$: # of unique keys in $p$ inserts
    \[
    |K| - \sum_{k \in K} (1 - f_X(k))^p
    \]
  - $\text{Unique}^{-1}(u)$: # of inserts for $u$ unique keys
  - $\text{Merge}(u, v)$: # of unique keys after merging $u$ and $v$ unique keys
    \[
    \text{Unique}(\text{Unique}^{-1}(u) + \text{Unique}^{-1}(v))
    \]

Accuracy & Speed of Our Method
- Only 3.8% error
- LevelDB-ana: Our LevelDB model
  - 0.01 sec/eval for 100 M unique keys (orders of magnitude faster)
- LevelDB-impl: Default level sizes (10X increase at each level)
- LevelDB-impl-opt: Optimized level sizes