Out-of-core Metadata for HDFS

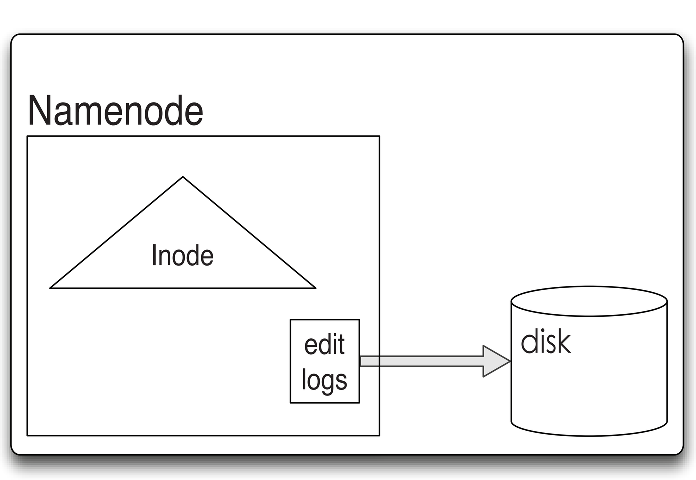
Lin Xiao, Garth Gibson (CMU)

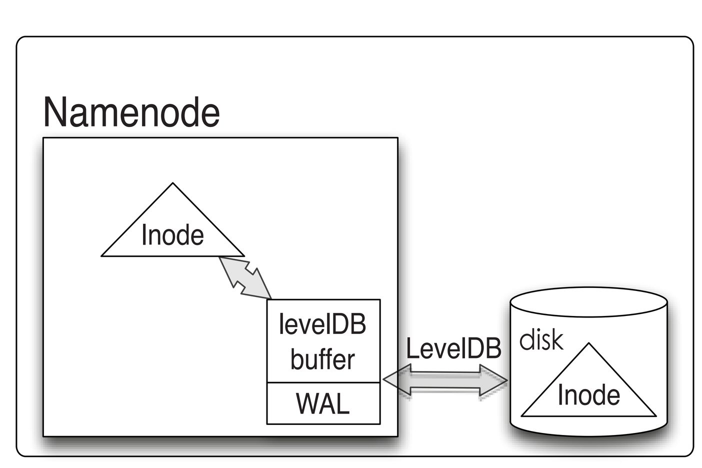
Motivation

- HDFS Namenode(NN) stores metadata in memory, and write-ahead logs and in infrequent checkpoint files
  - Design is simple
  - Metadata operations are fast
  - Maintains in-memory inodes in a tree structure
    - sub-sub-bullet
- Scaling metadata size separable from scaling thruput
  - Most files are small, suffer a high metadata footprint
  - Size of memory is the bottleneck
- Goal: remove space limits while maintain high performance of in-memory access when possible

Store Metadata in LevelDB

- LevelDB: key-value library uses log-structured merge tree on-disk
  - Local procedure calls to levelDB from HDFS NN
- Store namespace and block maps in LevelDB
  - All changes are written to LevelDB
  - No need to keep WAL in NN (b/c LevelDB has one)
- Other metadata is still in memory
  - Small and constantly changing
  - E.g. datanode status, leases, system stats





Cache Replacement Policy

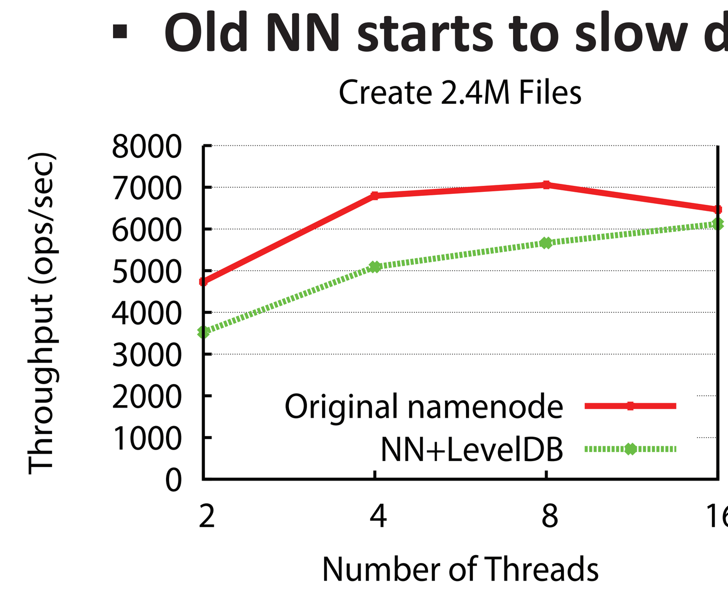
- Replacement on whole directories
  - Efficient write to disk
  - Reload whole directories on miss
  - Fast negative lookups for creating files
- Replacement policy: LRU
  - CLOCK to avoid excess bookkeeping overhead
  - No need to track every read for replacement policy
    - Only update last level directory
    - Visit /a/b/c updates /a/b
- A background thread monitors heap space
  - Eject directories when heap_size > threshold (e.g.90%)
  - Estimate/Monitor subtree size for later ejection
  - Move eviction out of locked critical sections

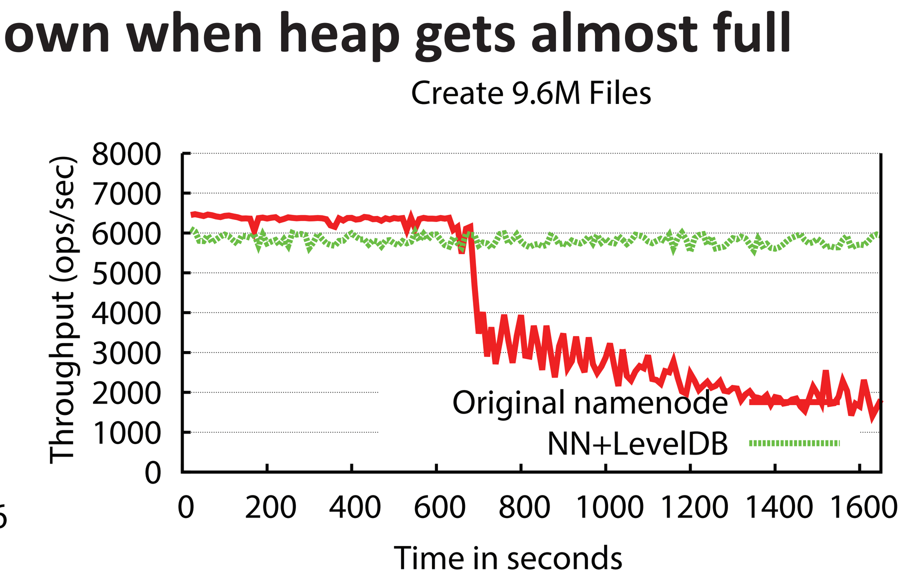
Overhead of Using LevelDB

- The effect of longer code path
  - Memory access → read from LevelDB
  - Metadata in memory + Write-ahead-log in LevelDB
- All changes are made persistent to levelDB
  - More writes than original namenode
- Group commit
  - No group commit in LevelDB
  - Batch updates to LevelDB as synchronous writes

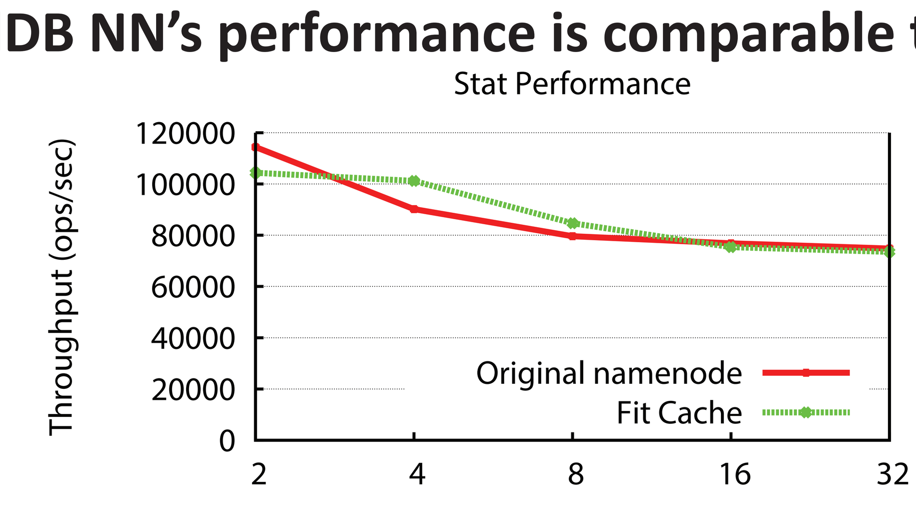
Performance

- Single Susitna node:
  - CPU: AMD Opteron 6272, 16-core 2.1 GHz
  - SSD: Crucial M4-CT064M4SSD2, 64 GB, SATA 6.0Gb/s
- NNThroughputBenchmark from Hadoop distribution
  - No RPC cost, run within NN & call FS methods directly
  - All operations are generated based on BFS order
- Create & close 2.4M files: all fit in cache
  - Old NN and LevelDB NN peak at different # threads
  - Degradation for peak throughput is 13.5%
- Create & close 9.6M files: 1% fits in cache
  - Old NN with 8 threads and LevelDB NN with 16 threads
  - Old NN starts to slow down when heap gets almost full





- Stat on first 600K of 2.4M files
  - Workload generator threads work on different part of NS
  - LevelDB NN's performance is comparable to original NN



Conclusions and Future Work

- Summer internship at Hortonworks
  - Jira HDFS-5389 for applying to HDFS
- Future workPartial directory fetch and ejection • Prefetching for better startup performance

Intel Science & Technology Center for Cloud Computing
Carnegie Mellon University
Georgia Tech
intel
Princeton University
UC Berkeley
University of Washington