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 throughput
  - 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

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

Conclusions and Future Work

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