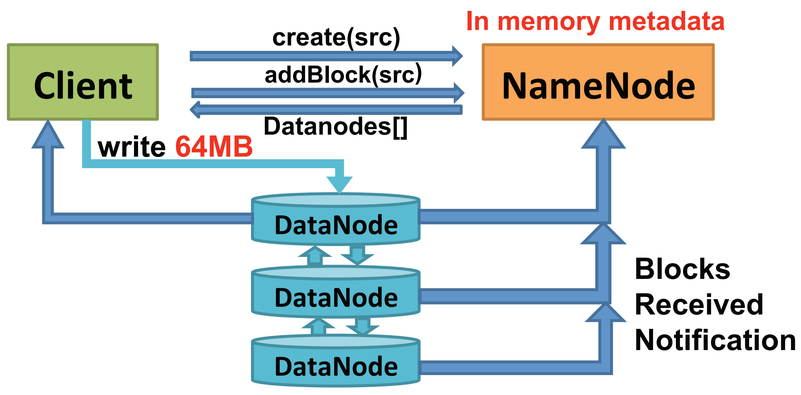
OVERCOMING METADATA BOTTLENECKS: SCALABLE HDFS

Lin Xiao, Wittawat Tantisiriroj, Garth Gibson (CMU)

SINGLE METADATA SERVER LIMITATIONS

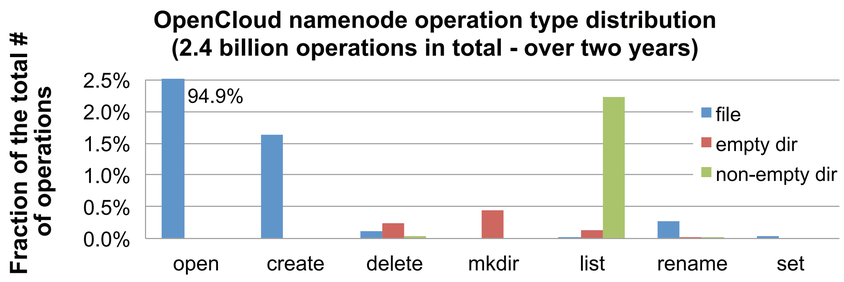


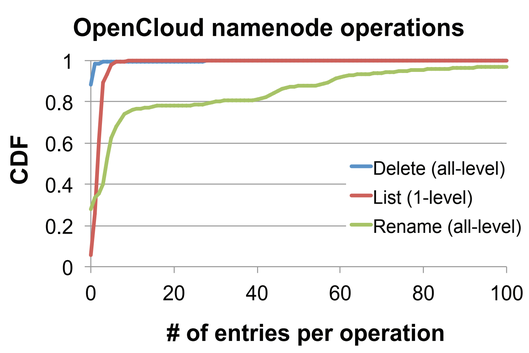
- Throughput limitations with one node
  - Metadata operation and block allocation
  - High rates of metadata operations for small files – intensive workloads
- Total number of files is limited by memory size
  - Follow GFS design to simplify and accelerate code path
  - Easy fix with disk data structures, but much lower throughput

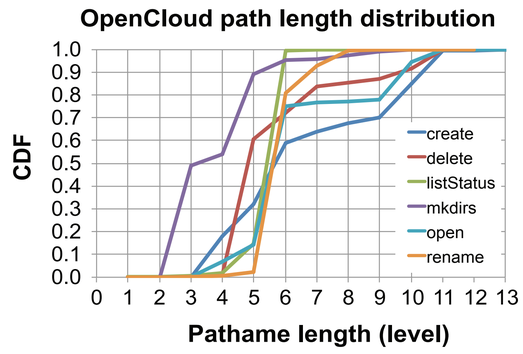
OUR GOALS

- Increase metadata operation throughput
  - Use multiple servers and avoid hot spot
- Low latency
  - When metadata fit in memory, as low as single server
- High availability and fault tolerance
- Explore tradeoffs for solving metadata bottleneck with table stores
  - Characterize table store features and implementations capable for scalable metadata service

OPENCLOUD METADATA OPERATION STATS







- open() is dominant operation
- Most list operations only involve a few files
- Path name length is mostly 5-10
  - May want to reduce iterative lookups for each path name component

WHY USE SCALABLE DB?

- Benefits:
  - Use multiple machines' CPU, memory, disks
  - Support backend storage with disks
  - Powerful and flexible DB code already exists
- Risks:
  - One RPC to NN may end up being many RPCs
    - Data in NN's memory will be spread out
  - Longer latency because of disk accesses
  - Longer code path with unneeded functionality

ROW KEY SCHEMAS

- DB fetch brings in multiple rows
  - So, what access patterns use the rest of each fetch?
- B-tree sorts on:
  - Full path name
    + one table lookup for each file
    - locality for whole subdirectory
    - rename a directory
  - Directory depth + full path name
    + locality for every directory
  - Parent Inode + file name (Inode + FN)
    - multiple lookups for each file
    + rename only changes inode
  - Hash(path name): Better load balance
    + load balanced

INTERESTING CHALLENGES

- Trade-offs in file metadata table schema
  - Optimization for group operations such as rename a directory could hurt a lookup operation
    - Indirection causes more lookups than full pathname
  - Maximizing locality leads to load imbalance
  - Use data duplication to optimize for different access patterns
- Columnar store
  - Locality group based on access pattern
  - Reduce clean up work such as major compactions
- Reduce latency caused by distributing metadata
  - Collocate processes with tablet server
    - Coprocessor in HBase, iterator in Accumulo
  - Send requests in parallel
  - Aggressive caching is feasible if clients can tolerate temporary outdated data
- Efficient use of memory
  - Memory overhead compared to customized service

Intel Science & Technology Center for Cloud Computing

Carnegie Mellon University · Georgia Tech · intel · Princeton University · UC Berkeley