STORAGE DEVELOPER CONFERENCE
SDC 21
BY Developers FOR Developers
Virtual Conference
September 28-29, 2021
A SNIA Event

Compacting small objects in cloud for high yield

Tejas Chopra, Senior Software Engineer, Netflix, Inc.

Agenda

- About me
- Problem statement & overview
- Compaction basics
- Data structures: blobs, blob tables
- File system operations
- Results
- Conclusion

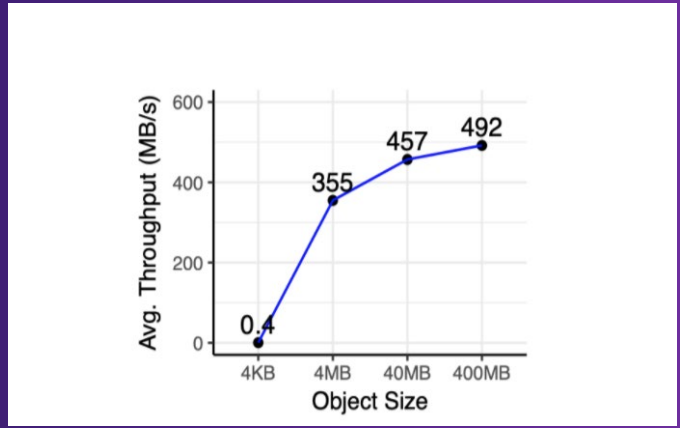
About me

- Senior Software Engineer, Netflix
- Keynote speaker: Distributed systems, Cloud, Blockchain
- Senior Software Engineer, Box
- Datrium, Samsung, Cadence, Tensilica

NETFLIX

Overview

- Cloud FS: Provide FS interface to objects
  - Simple approach: Each file → object
  - Cloud workloads: Many small files, bursty
- Small sized files, not beneficial for Cloud Storage
- Request Level costs
  - To upload 1GiB file with 4KiB PUTs is 57x the cost of storing the file
- Throttling
  - S3 Best Practice Guideline: If > 300 PUT/DELETE/LIST/s or > 800 GET/s, S3 may rate limit
  - 1 4MiB PUT is 1000x faster than 1000 4KiB PUTs



Cloud File system options

- EFS: Generic File system: Charges only for storage: $0.3/GB/month
- DynamoDB: NoSQL, $0.25/GB/month for 1 write/s/month. $0.09/GB/month for 1 read/s/month
- S3: $0.023/GB/month, 0.0005c/PUT
- To store 1 TiB worth of 4KiB files, 100 writes/s:
  - S3 (as is): $1366; operation cost: 98%; 5 yrs of storage to match operation cost
  - EFS: $307, operation cost: 0%
  - DynamoDB: $303, operation cost: 16%; 5 days of storage to match operation cost
  - S3 (with compaction): $24, operation cost: 0.001%; 27 seconds of storage to match operation cost.

Compaction

- Pluggable module on the client side
- Module packs data into GiB sized blobs and an index of where the file is within the blob
- Redundant client side global index of all files in all blobs
- Embedded index in every blob is for fault-tolerance

Data Structures: Blob

- Single, immutable object on cloud
- Data from multiple files is packed into a blob. Footer contains information about the byte ranges of a file within the blob
- By reading all footers of all compacted blobs, we can recreate the world
- Could have used SSTables, but it requires sorting before persistence

Data Structures: Blob Table

- Optimization over reading all blob footers
- Global table that contains information of all files in all blobs
- Redundant - can be recreated
- Consistency is a challenge because Blob creation is distributed.

Compaction

- Data is buffered on client nodes, and is not durable until pushed to cloud
- Compaction policy is specified at mount time
- Once data is compacted, embedded indices within the blob get updated
- Blob name, objectId: <clientIP>:<FooterOffset>:<Date>
- After blob is pushed to cloud, blob table is updated with new extents for a file
- Global blob table always maintains the updated loc for each file

Reading Data

- Depends on whether data is buffered or pushed to cloud
- Blob Table contains the location of data
  - If on a client node, client-client transfer for read
- Range reads for cloud blobs
  - May want to fetch more than just what's needed - essence of prefetching

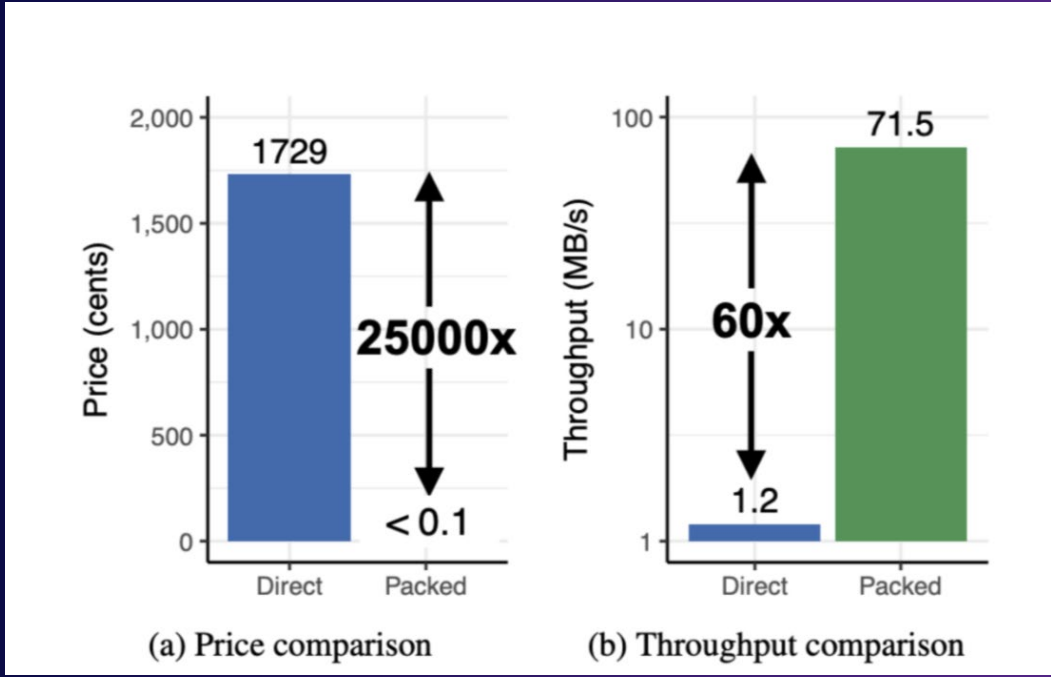
Deletes and Renames

- Deletes
  - When client issues a delete, remove entry from Blob table
  - Periodically check the liveness of a blob
  - If live files are less than a threshold, repack the blob
- Renames
  - Atomic update of file's name in Blob table
  - Piggyback on future blobs' embedded indices
  - For recovery from embedded indices, blobs are read in order of creation, so latest blobs will contain the rename information

Fault Tolerance

- Master
  - Periodically backup global Blob table to cloud
  - On a crash, read last checkpointed Blob table and all the blobs written after that time
- Clients
  - Files buffered for compaction are lost
  - Can be still recovered if persisted on the local disk
  - If compacted and pushed to cloud, Blob table contains all information for recovery
  - If compacted and pushed to cloud, but the Blob table not updated, the embedded indices contain information for recovery

Results



Conclusion

- Cloud file systems should aggressively compact small objects
- Significant costs for operations and rate limiting
- Throughput increased by 60x, costs reduced by 25000x
- Backups of smaller files can be archived using this strategy to improve backup times and costs
- Compacting policies can compact objects by create-time, per-application, etc. We could apply learning to choose objects to compact, objects with similar read profiles can benefit from prefetching the entire blob

Thank you!

https://www.linkedin.com/in/chopratejas

chopratejas@gmail.com

chopra_tejas

