Secondary Index Storage Plasma - Design to Scale

Presented by Durga Akhil Mundroy

SESSION FOCUS: Global Secondary Index

Agenda

01/ Plasma Design

02/ Memory Management

03/ Log File Management

04/ Scaling

05/ I/O Improvements and Recovery

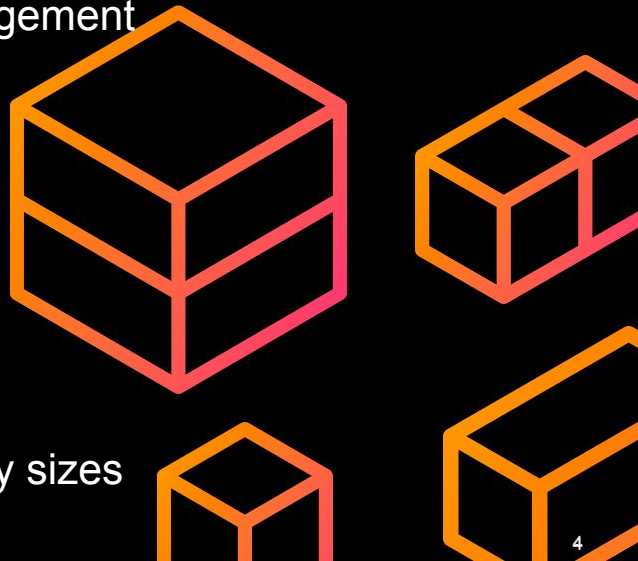
06/ Stats

1 Plasma Design

Plasma Design

Architecture Overview

- Storage Engine for Secondary Indexing for Range Scan and Point Lookup
- Proprietary lock-free data structure enables highly concurrent index updates
- Use snapshot to eliminate locking during concurrent range scan
- Least-Recently-Used like eviction algorithm for memory management
- Concurrent design enables high throughput on scan workload with high locality
- Use append-only Log Structured Persistent Storage (LSS) optimized for SSD/NVME
- Support Incremental Compaction on LSS
- Use checkpointing for fast recovery
- Auto tune based on workload, available memory and index key sizes

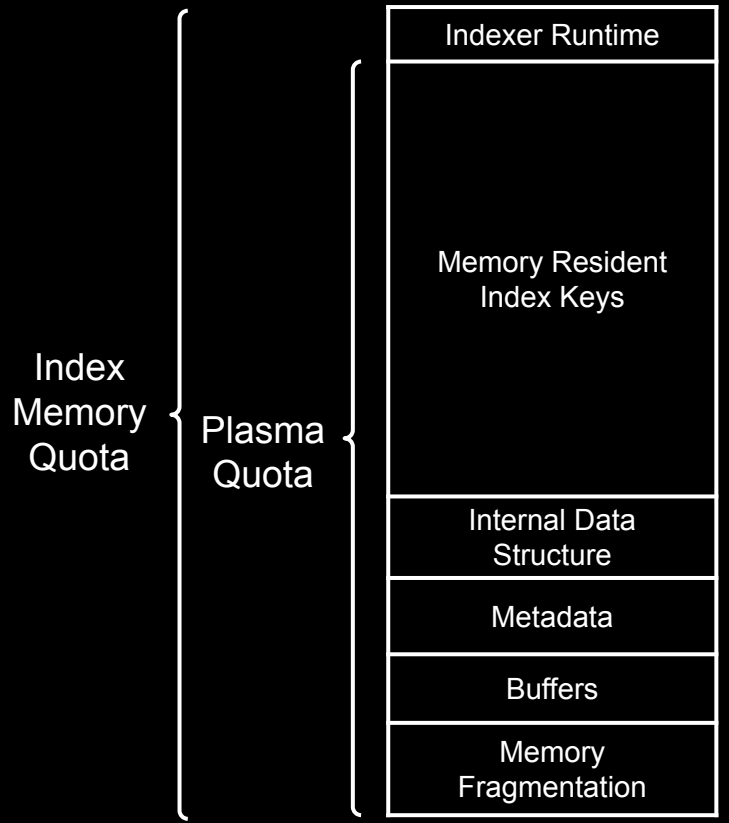


2 Memory Management

Memory Management

Plasma Memory Quota

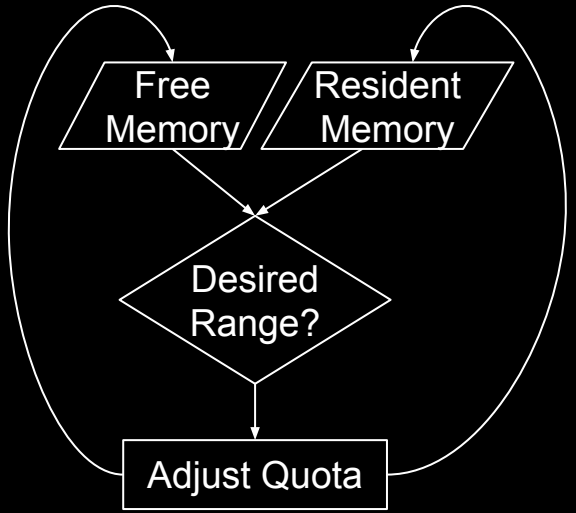
- Memory usage comes from:
  - Cached index keys
  - Internal data structure
  - Metadata
  - Buffers
  - Internal memory fragmentation
- Quota limits maximum memory use for index keys
- Honor quota on best effort basis
- Evict data from memory to adhere to the quota
  - Cause index resident percent to reduce
- Plasma uses 90% of index memory quota
- Need headroom between quota and system memory



Memory Management

Memory Quota Tuner

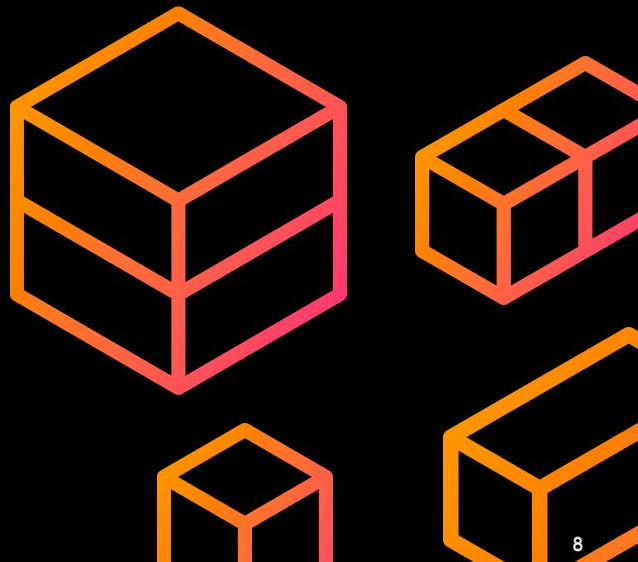
- Dynamically adjust plasma quota to create more headroom
- Counteract memory use by internal memory fragmentation and other processes external to indexer to avoid out-of-memory scenario
- Tuning algorithm looks at system metrics and is based on Feedback Control Systems
  - System free memory
  - Indexer process resident memory
- Decrement if less free memory or resident memory is too high
  - Create headroom by evicting memory resident data
  - Purge freed memory to OS
- Increment if enough headroom



Memory Management

Eviction

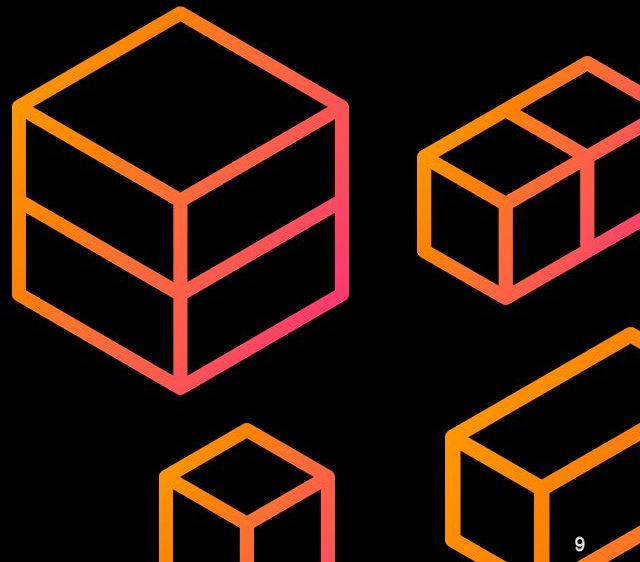
- Eviction will free memory resident index keys
- During eviction, any dirty index keys will be flushed to disk
- Evict index keys that have not been accessed recently
- Triggers for eviction
  - Swapper - background threads that evict pages
    - Burst Eviction - runs only when memory usage is greater than quota and evicts to alleviate memory pressure
    - Periodic Eviction - runs continuously in the background, sweeping index keys to determine and maintain working set in memory
  - Eviction during scans and mutations if they throttle under memory pressure



Memory Management

Swapper - Periodic Eviction

- Determine working set of data to keep in memory
- Treat data accessed in the last 10 minutes as the working set - called sweep interval
- If determined working set does not fit in quota, proportionally shorten the sweep interval
  - Retain most recently accessed data to exploit temporal locality
- If determined working set fits in quota, preemptively evict to create headroom for sudden burst of traffic
  - Evict till memory usage is 50% of quota
  - Evict only if cannot maintain 100% resident ratio

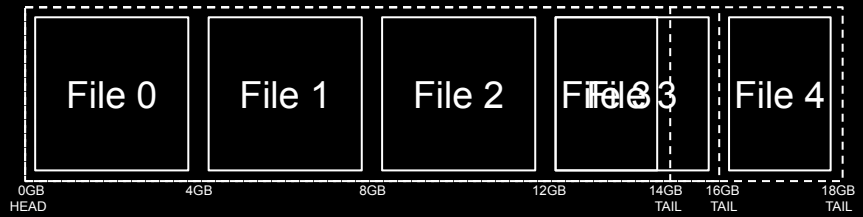


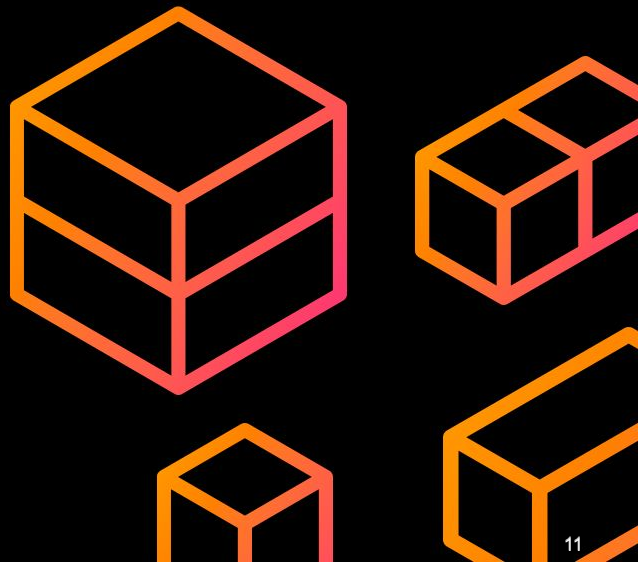
3 Log File Management

Log File Management

Plasma LSS - Log Structured Store - Logical Log

- A contiguous logical 64-bit log space
- Log Head - offset into first byte of data in LSS
- Log Tail - offset into last byte of data in LSS
- Broken into a list of 4GB physical files
- Each file maps to an offset range in logical log space
- As new data is written (append only), log space grows (Log Tail advances)
  - New physical files added when needed

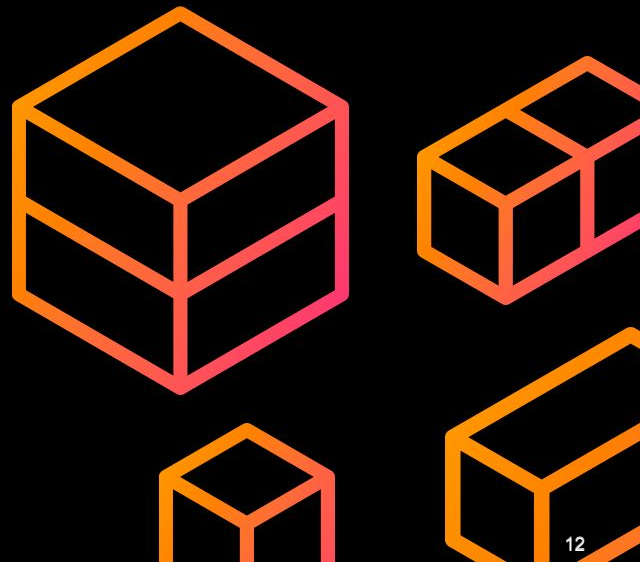




Log File Management

Disk Fragmentation

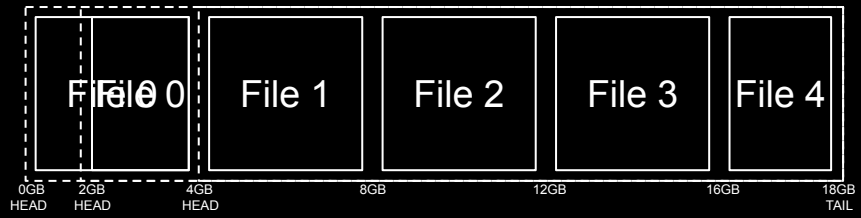
- Plasma manages index keys in ranges called index page
- As new data is written, an index page may get rewritten
  - A new index page is appended to LSS
  - Existing index page on file can become stale
- An index page can be rewritten to speed up index scan
  - Page Compaction
  - Full page persistence
- Fragmentation ratio is the fraction of stale data on disk
  - Fragmentation depends on mutation rate
  - More new data (higher mutation rate), more data can become stale



Log File Management

Log Cleaner - Incremental Log Compaction

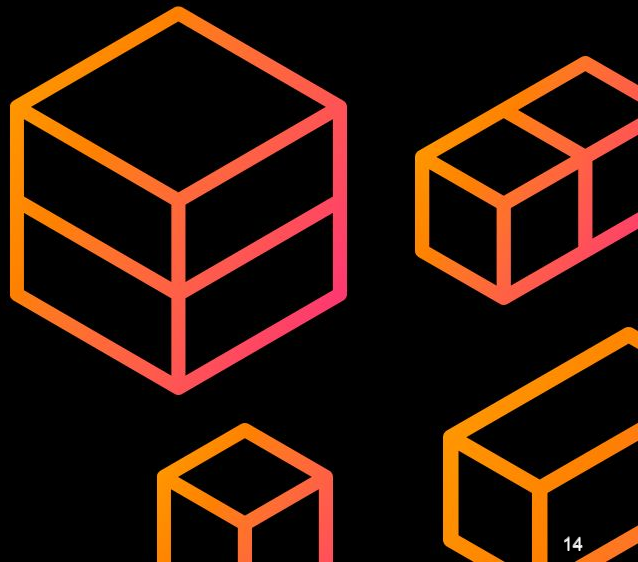
- Log Cleaner - read index pages in LSS from Log Head to Log Tail
  - If index page is stale, remove from LSS, advance Log Head
  - If index page is still valid, append to LSS, advance Log Head
- As LSS is compacted
  - If physical file still has some valid data - shrink file (hole punch)
  - If physical file no longer has any valid data - delete file
- Disk Compaction is incremental
  - Maintain 30% fragmentation
  - Cost is proportional to mutation rate
- If fragmentation is over 80%, throttle write throughput to avoid running out of disk space



Log File Management

Plasma Log Cleaner - Hole Punching

- Hole Punch - reduce file size without deleting the whole file
- Only if physical file has some valid data after disk compaction
- At least 64MB of contiguous stale data
- Only on supported file systems - check at startup
- May not be supported on network and encrypted file systems

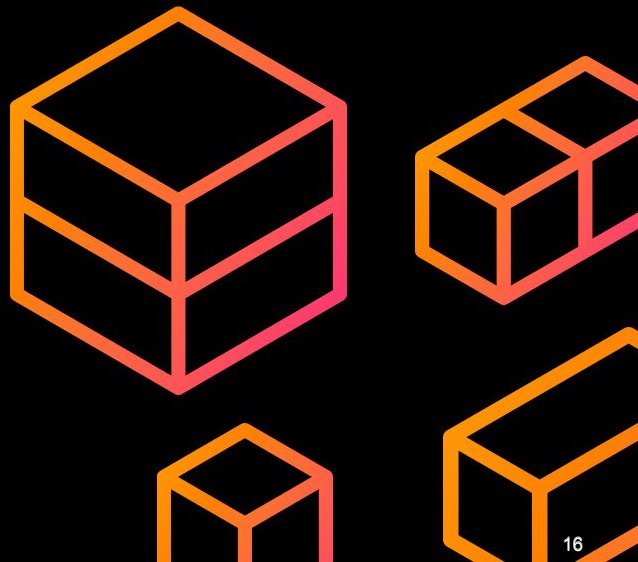


4 Scaling

Scaling to 10K indexes

Challenges

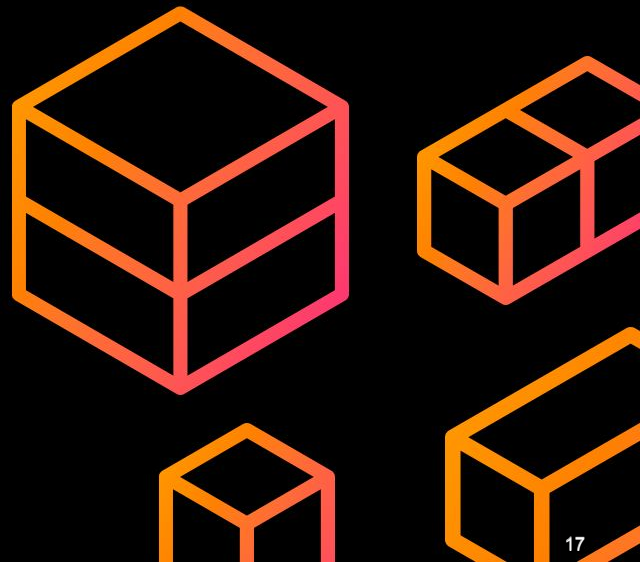
- Each index runs its own eviction algorithm
  - A “hot” index cannot evict an unused page from “cold” index
- Each index allocates its own buffers
  - High memory overhead with large number of indexes
- Each index has a dedicated set of “threads” (goroutines)
  - High CPU overhead with large number of indexes
  - Can hit thread limit per process
- Each index has its own log structured storage (LSS)
  - Large number of files with large number of indexes
  - High I/O overhead for log maintenance
  - Degrade I/O performance with high concurrent traffic



Scaling to 10K indexes

Solution

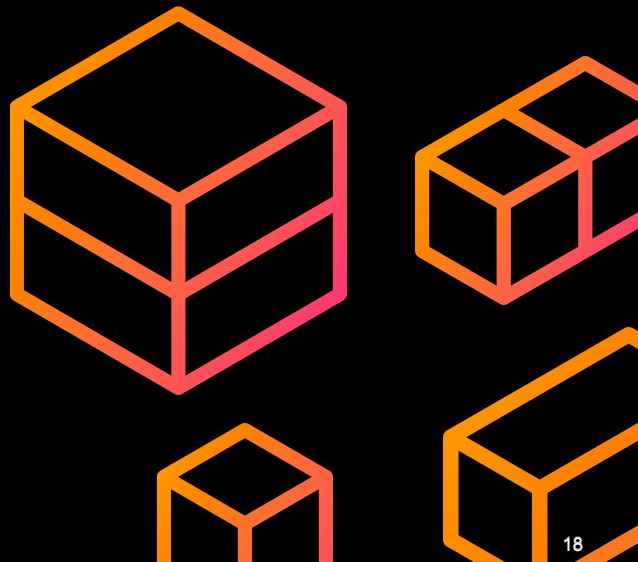
- Scaling requires efficient resource management across indexes
- Share resources across indexes
  - Resource can be assigned on-the-fly to where demand is needed
  - Resource can be reclaimed on-the-fly from where demand has subsided
  - Resource allocation can be elastic
- Improve resource utilization and efficiency
  - Resource overhead cannot be proportional to index size
  - Resource overhead cannot be proportional to number of indexes
  - Reduce overall memory and I/O overhead
- GSI can support 10K indexes on collections in 7.0



Scaling to 10K indexes

Sharding - Resource Sharing

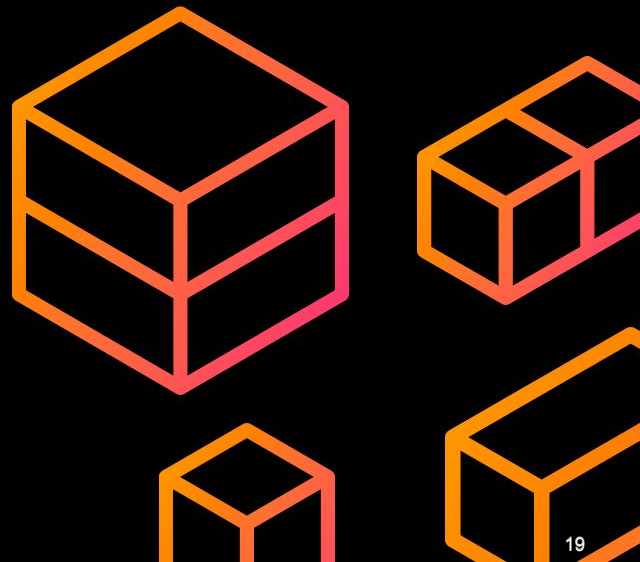
- Indexes are grouped into shards
- Each shard is a unit of resource management
  - Share swapper for eviction
  - Share background workers (threads/goroutines)
  - Share buffer pool
  - Share LSS (Collection Index Only)



Scaling to 10K indexes

Improve Resource Efficiency

- Many enhancements intended for improving efficiency in resource utilization
  - Bloom Filter
  - Periodic Eviction and Self-tuning Sweep Interval
  - Opportunistic Page Compaction
  - Incremental Checkpoint

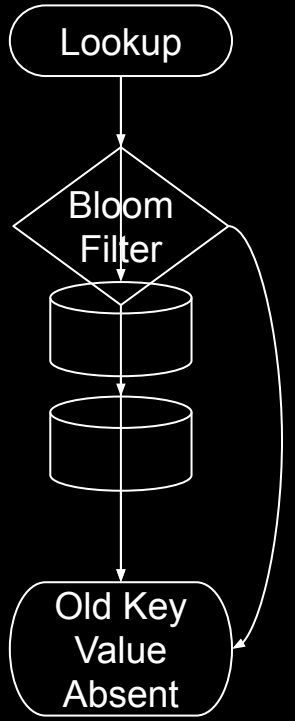


5 I/O Improvements and Recovery

I/O Improvements

Bloom Filter - Insert Heavy Workloads

- Processing mutation requires two steps
  - Lookup and Delete old index key value
  - Insert new index key value
- Insert new index key - no need to lookup for old index key value
- Use Bloom Filter to track key value on disk
- Bloom Filter is a space-efficient probabilistic Set data structure
- If old index key value not in Bloom filter, skip lookup old key value from disk
- Reduce random I/O for lookup of old key for insert heavy workload
- Memory usage depends on number of items, resident ratio and configured accuracy of bloom filter
  - Disabled by default and can be turned on and tuned using indexer setting
  - For every item on disk, bloom filters consume ~5 bits



I/O Improvements

Opportunistic Page Compaction

- Plasma is MVCC (multi-version concurrency control) for supporting index snapshot
- An index can contain old values, new values and deleted index keys
- Old/deleted key values need to be compacted away in index page when no longer in use
  - Preserve memory and disk space
  - Reduce I/O needed to read and write page to disk
- Before 7.0, page compaction done when page accumulates N mutations
- Can incur additional I/O for fetching page into memory from disk
- In 7.0.2, page compaction is opportunistic without incurring additional I/O
  - When a page is split into two because the page has become full
  - When a frequently modified page is written to disk
  - When a page is rewritten during incremental disk compaction by log cleaner
  - When a page with a lot of old or deleted values is being scanned on (7.1)

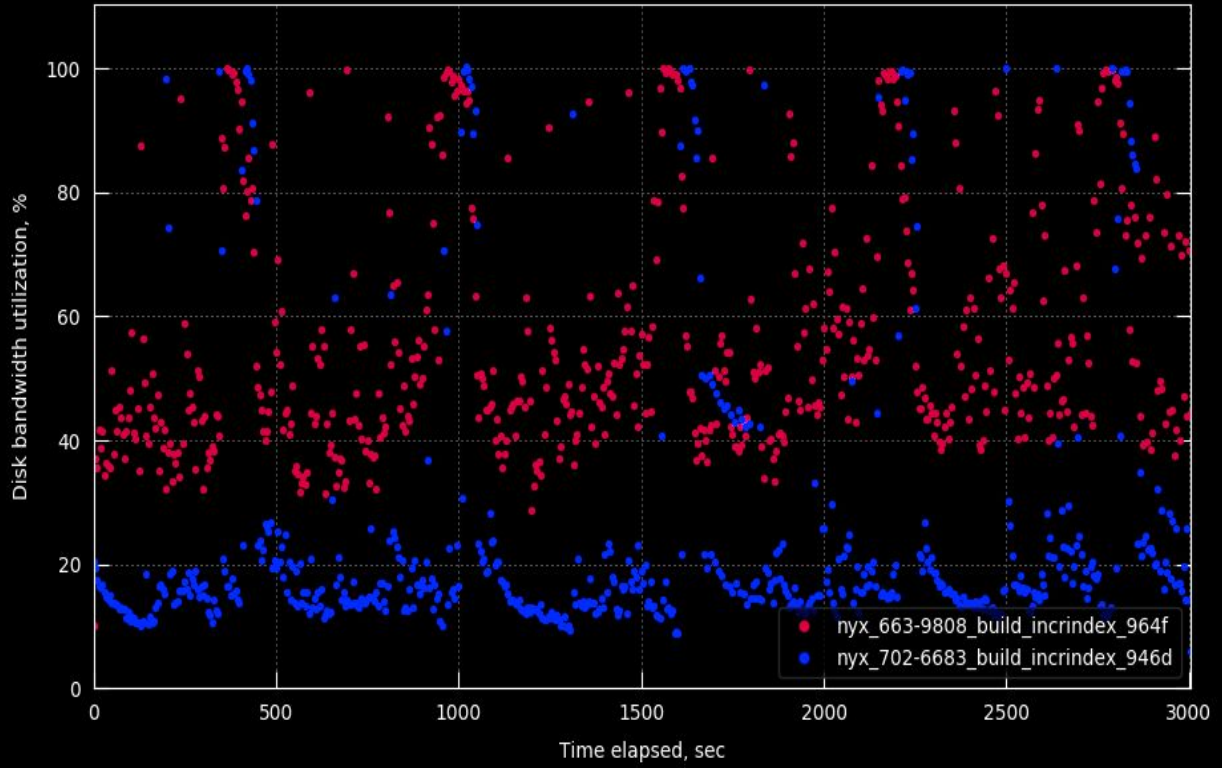
I/O Improvements

Incremental Checkpoint

- Indexer creates persistent checkpoint every 10 minutes for crash recovery
- Checkpoint saves index metadata for fast recovery
- Full checkpoint (pre-7.0) persists
  - New mutations since previous checkpoint (cost proportional to mutation rate)
  - Metadata for all index pages (cost proportional to index size)
- Incremental checkpoint (7.0) improves by persisting only metadata of dirty index pages since previous checkpoint (cost proportional to mutation rate)
- Reduce CPU and I/O usage with large number indexes (especially “cold” indexes)
- All incremental checkpoints are saved to a log structured storage (append-only)
- Recovery involves reading index metadata from all incremental checkpoints and restoring the most recent metadata

I/O Improvements

Results - Disk I/O during incremental load



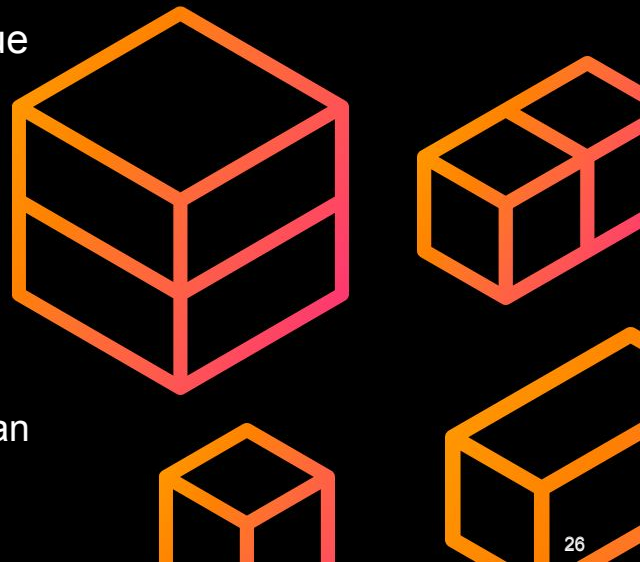
- Disk Utilization improvement
  - Lower is better
- Red is 6.6.3
- Blue is 7.0.2

6 Stats

Stats

Index Stats

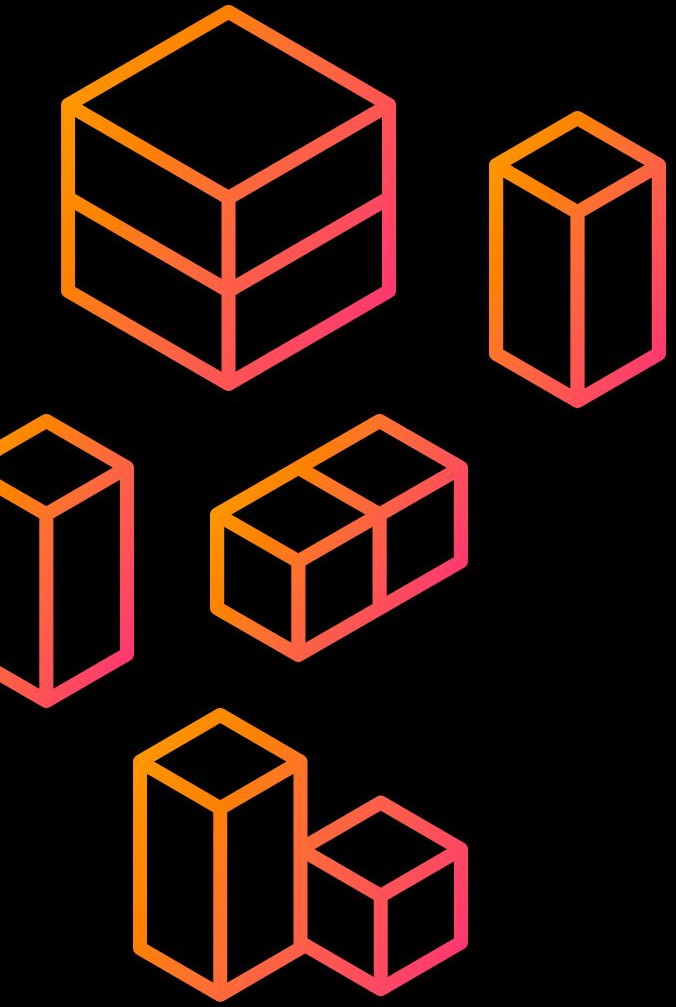
- Resident Ratio
  - Fraction of index keys present in memory
  - Depends on memory quota and working set
  - 20% is minimum recommended resident ratio
  - If goes too low, mutation and scan processing can slow down due to increased disk I/O. Must revisit sizing to increase the quota
- Index Cache Miss Ratio
  - Fraction of needed index keys that were not in memory
  - Measure of effectiveness of plasma caching
  - If high, indicates that more disk I/O was needed and that the working set does not fit in memory
    - Must revisit the queries or sizing so that more of the working set can be cached



Stats

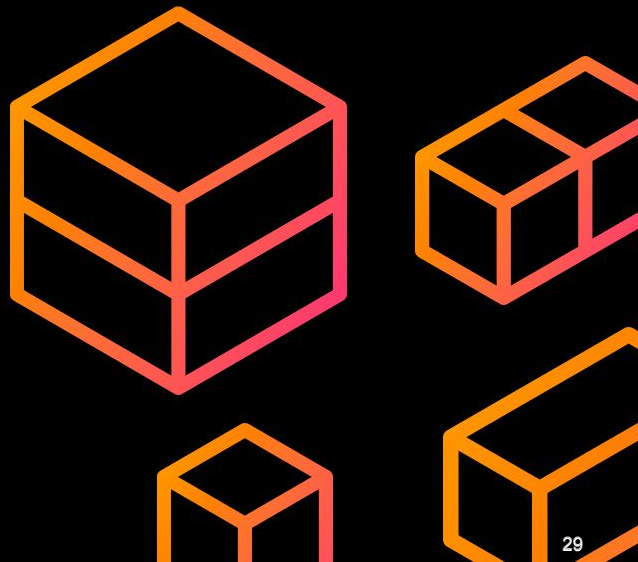
Index Stats

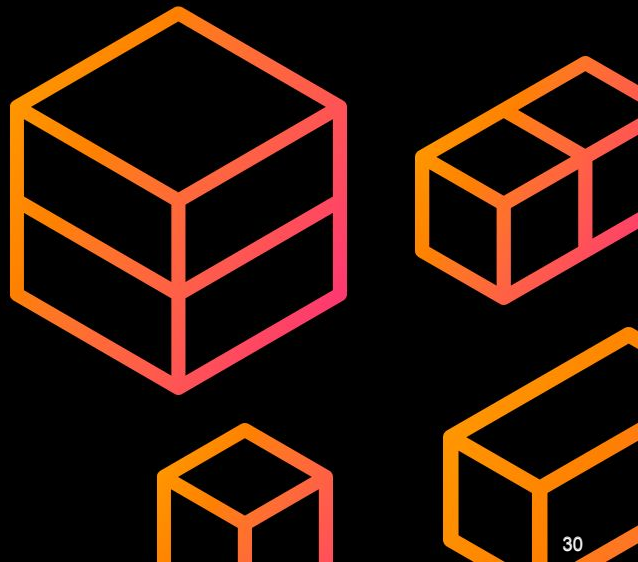
- Index Data Size
  - Size of data in the index, uncompressed, not including metadata and overheads
- Index Disk Size
  - Size of data on disk, compressed, including metadata and overheads like stale data
  - If disk size is large compared to data size, check if hole punching is supported and if fragmentation is being kept in check
- Index Fragmentation
  - Percentage of stale data on disk
  - Ideally close to 30%. If close to 80%, then the mutation rate can be affected
  - Higher resident ratio leaves more disk bandwidth for disk compaction to keep fragmentation in check

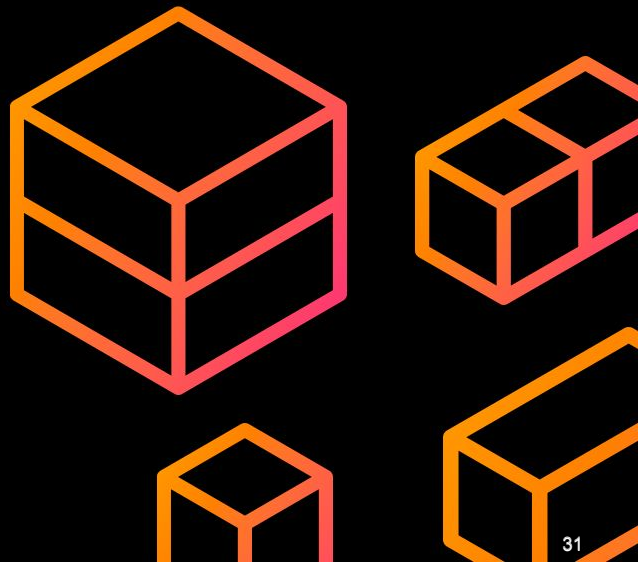


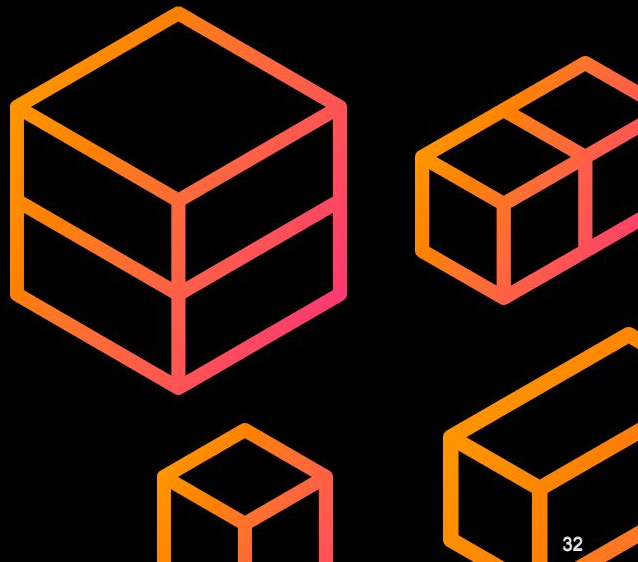
THANK YOU

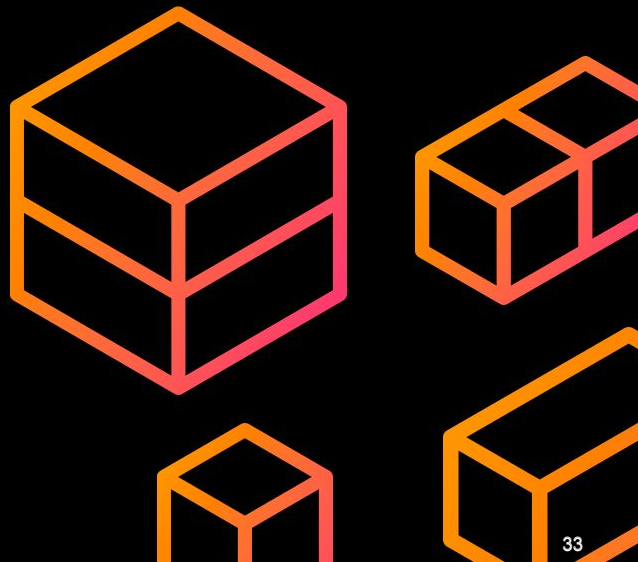
Couchbase











Agenda

01/ Plasma Design

02/ Memory Management

03/ Log File Management

04/ Scaling

05/ I/O Improvements and Recovery