SCYLLA'S COMPACTION STRATEGIES

OR

HOW TO RUIN YOUR WORKLOAD'S PERFORMANCE BY CHOOSING THE WRONG COMPACTION STRATEGY

Nadav Har'El, Raphael Carvalho

SCYLLA.

Nadav Har'El

SCYLLA.

Nadav Har'El has had a diverse 20-year career in computer programming and computer science. In the past he worked on scientific computing, networking software, and information retrieval. In recent years his focus has been on virtualization and operating systems. He also worked on nested virtualization and exit-less I/O in KVM. Today, he maintains the OSv kernel and also works on Seastar and Scylla.

Raphael Carvalho

SCYLLA.

Raphael S. Carvalho is a computer programmer who loves file systems and has developed a huge interest in distributed systems since he started working on Scylla. Previously, he worked on ZFS support for OSv and also drivers for the Syslinux project. At ScyllaDB, Raphael has been mostly working on compaction and compaction strategies.

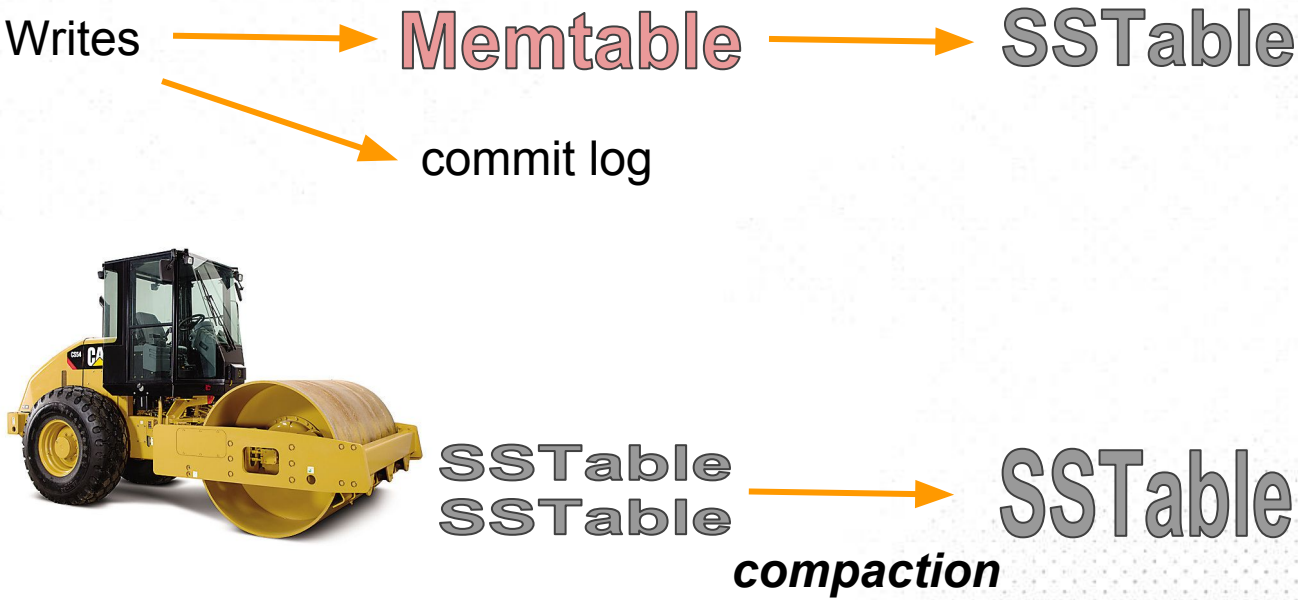
Agenda

- What is compaction?
- Scylla's compaction strategies:
  - Size Tier
  - Leveled
  - **Hybrid** 

  - Date Tier
  - Time Window
- Which should I use for my workload and why?
- Examples!

What is compaction?

Scylla's write path:

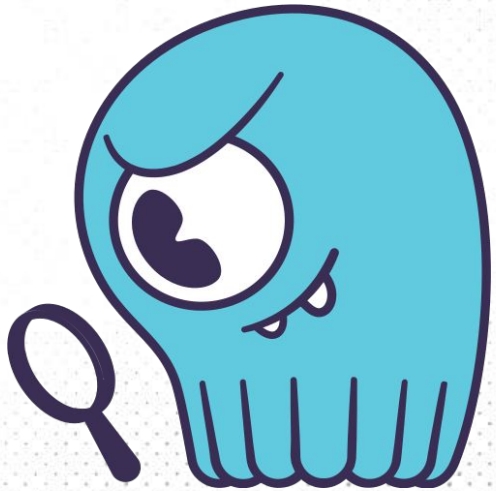


(What is compaction?)

- Scylla's write path:
  - Updates are inserted into a memory table ("memtable")
  - Memtables are periodically flushed to a new sorted file ("sstable")
- After a while, we have many separate sstables
  - Different sstables may contain old and new values of the same cell
  - Or different rows in the same partition
  - **Wastes disk space**
  - **Slows down reads**
- **Compaction**: read several sstables and output one (or more) containing the merged and most recent information

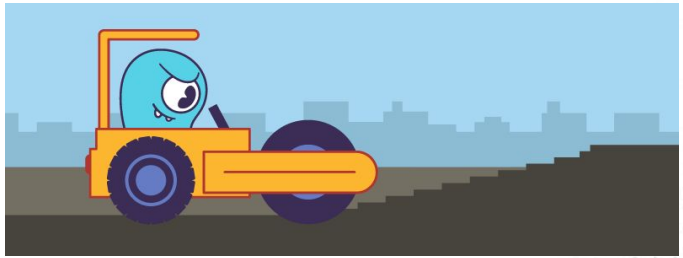
What is compaction? (cont.)

- This technique of keeping sorted files and merging them is well-known and often called **Log-Structured Merge** (LSM) **Tree**
- Published in 1996, earliest popular application that I know of is the Lucene search engine, 1999
  - High performance write.
  - Immediately readable.
  - Reasonable performance for read.



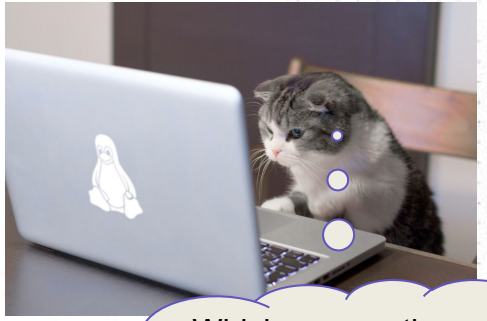
(Compaction efficiency requirements)

- Sstable merge is efficient
  - Merging sorted sstables efficient, and contiguous I/O for read and write
- Background compaction does not increase request tail-latency
  - Scylla breaks compaction work into small pieces
- Background compaction does not fluctuate request throughput
  - “Workload conditioning”: compaction done not faster than needed



Compaction Strategy

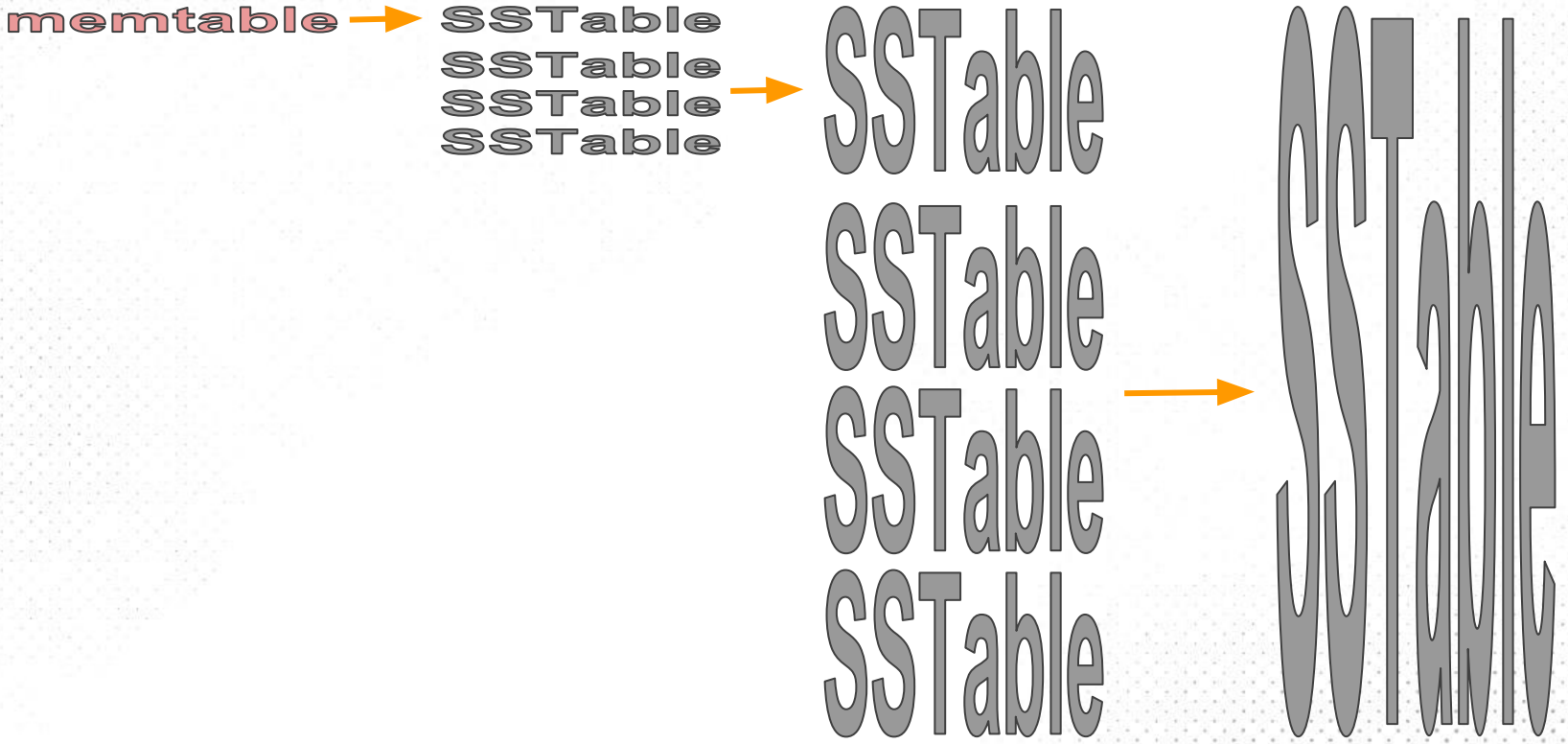
- Which sstables to compact, and when?
- This is called the **compaction strategy**
- The goal of the strategy is low amplification:
  - Avoid read requests needing many sstables.
    - **read amplification**
  - Avoid overwritten/deleted/expired data staying on disk.
  - Avoid excessive temporary disk space needs (scary!)
    - **space amplification**
  - Avoid compacting the same data again and again.
    - **write amplification**



Strategy #1: Size-Tiered Compaction

- Cassandra's oldest, and still default, compaction strategy
- Dates back to Google's BigTable paper (2006)
  - Idea used even earlier (e.g., Lucene, 1999)

Size-Tiered compaction strategy



(Size-Tiered compaction strategy)

- Each time when enough data is in the memory table, flush it to a small sstable
- When several small sstables exist, compact them into one bigger sstable
- When several bigger sstables exist, compact them into one very big sstable
- …
- Each time one “size tier” has enough sstables, compact them into one sstable in the (usually) next size tier

Size-Tiered compaction - amplification

- **write amplification**: O(logN)
  - Where “N” is (data size) / (flushed sstable size).
  - Most data is in highest tier - needed to pass through O(logN) tiers
  - This is asymptotically optimal

Size-Tiered compaction - amplification

What is **read amplification?** O(logN) sstables, but:

- If workload writes a partition once and never modifies it:
  - Eventually each partition's data will be compacted into one sstable
  - In-memory *bloom filter* will usually allow reading only **one** sstable
  - Optimal
- But if workload continues to update a partition:
  - All sstables will contain updates to the same partition
  - O(logN) reads per read request
  - Reasonable, but not great ☹

Size-Tiered compaction - amplification

- **Space amplification** ☹

DISK SPACE. THE FINAL FRONTIER.

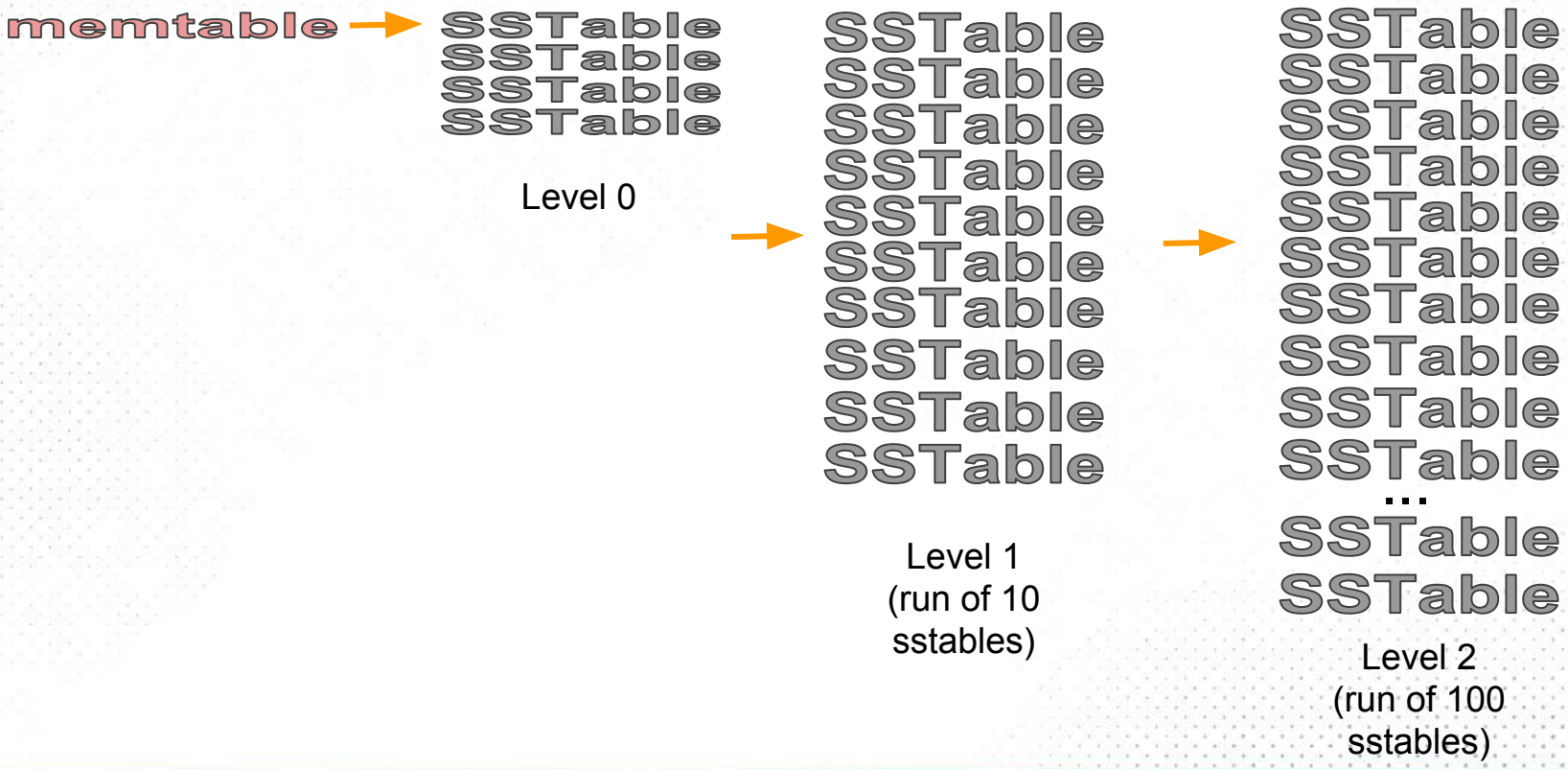
Size-Tiered compaction - amplification

- **Space amplification**:
  - Obsolete data in a huge sstable will remain for a very long time
  - Compaction needs a lot of temporary space:
    - Worst-case, needs to merge all existing sstables into one and may need **half** the disk to be empty for the merged result. (2x)
    - Less of a problem in Scylla than Cassandra because of sharding
  - When workload is overwrite-intensive, it is even worse:
    - We wait until 4 large sstables
    - All 4 overwrote the same data, so merged amount is same as in 1 sstable
    - 5-fold space amplification!
    - Or worse - if compaction is behind, there will be the same data in several tiers and have unequal sizes

Strategy #2: Leveled Compaction

- Introduced in Cassandra 1.0, in 2011.
- Based on Google's LevelDB (itself based on Google's BigTable)
- No longer has size-tiered's huge sstables
- Instead have **runs**:
  - A **run** is a collection of small (160 MB by default) SSTables
  - Have non-overlapping key ranges
  - A huge SSTable must be rewritten as a whole, but in a run we can modify only parts of it (individual sstables) while keeping the disjoint key requirement
- In leveled compaction strategy:

Leveled compaction strategy



(Leveled compaction strategy)

- SSTables are divided into “levels”:
  - New SSTables (dumped from memtables) are created in Level 0
  - Each other level is a run of SSTables of exponentially increasing size:
    - Level 1 is a run of 10 SSTables (of 160 MB each)
    - Level 2 is a run of 100 SSTables (of 160 MB each)
    - etc.
- When we have enough (e.g., 4) sstables in Level 0, we compact them with all 10 sstables in Level 1
  - We don't create one large sstable - rather, a run: we write one sstable and when we reach the size limit (160 MB), we start a new sstable

(Leveled compaction strategy)

- After the compaction of level 0 into level 1, level 1 may have more than 10 of sstables. We pick one and compact it into level 2:
  - Take one sstable from level 1
  - Look at its key range and find all sstables in level 2 which overlap with it
  - Typically, there are about 12 of these
    - The level 1 sstable spans roughly 1/10th of the keys, while each level 2 sstable spans 1/100th of the keys; so a level-1 sstable's range roughly overlaps 10 level-2 sstables plus two more on the edges
  - As before, we compact the one sstable from level 1 and the 12 sstables from level 2 and replace all of those with new sstables in level 2

(Leveled compaction strategy)

- After this compaction of level 1 into level 2, now we can have excess sstables in level 2 so we merge them into level 3. Again, one sstable from level 2 will need to be compacted with around 10 sstables from level 3.

Leveled compaction - amplification

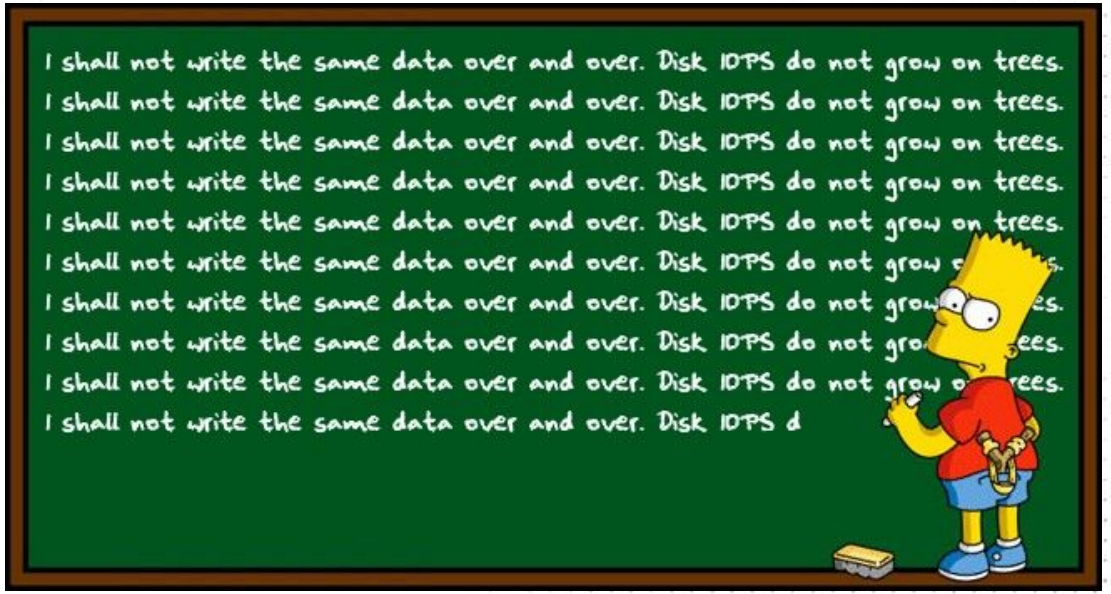
- **Space amplification:**
  - Because of sstable counts, 90% of the data is in the deepest level (if full!)
  - These sstables do not overlap, so it can't have duplicate data!
  - So at most, 10% of the space is wasted
  - Also, each compaction needs a constant (~12*160MB) temporary space
  - Nearly optimal

Leveled compaction - amplification

- **Read amplification:**
  - We have O(N) tables!
  - But in each level sstables have disjoint ranges (cached in memory)
  - Worst-case, O(logN) sstables relevant to a partition - plus L0 size.
  - Under some assumptions (update complete rows, of similar sizes) space amplification implies: 90% of the reads will need just one sstable!
  - Nearly optimal

Leveled compaction - amplification

- **Write amplification**: ☹



Leveled compaction - amplification

- **Write amplification**:
  - Again, most of the data is in the deepest level k
    - E.g., k=3 is enough for 160 GB of data (per shard!)
    - All data was written once in L0, then compacted into L1, … then to Lk
    - So each row written k+1 times
  - For each input (level i>1) sstable we compact, we compact it with ~12 overlapping sstables in level i+1. Writing ~13 output sstables. (lower for L0)
  - Worst-case, write amplification is around 13k
  - Also O(logN) but higher constant factor than size-tiered...
  - If **enough writing and LCS can't keep up**, its read and space advantages are lost
  - If also have cache-miss reads, they will get less disk bandwidth

Example 1 - write-only workload

- Write-only workload
  - Cassandra-stress writing 30 million partitions (about 9 GB of data)
  - Constant write rate 10,000 writes/second
  - One shard

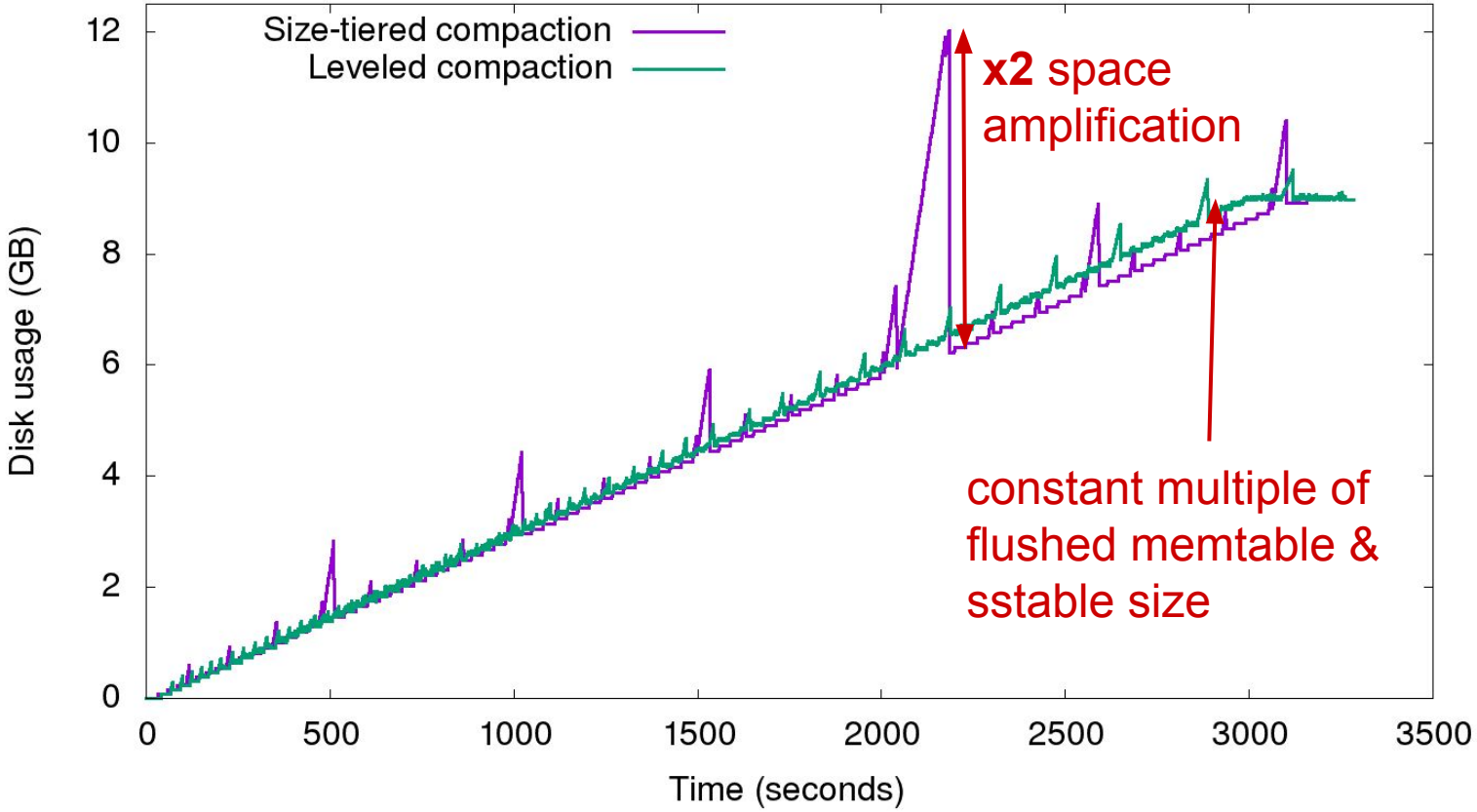
Example 1 - write-only workload

- Size-tiered compaction:

  at some points needs twice the disk space
  - In Scylla with many shards, “usually” maximum space use is not concurrent
- Level-tiered compaction:

  more than double the amount of disk I/O
  - Test used smaller-than default sstables (10 MB) to illustrate the problem
  - Same problem with default sstable size (160 MB) - with larger workloads

Example 1 (space amplification)



Example 1 (write amplification)

- Amount of actual data collected: 8.8 GB
- Size-tiered compaction: 50 GB writes (4 tiers + commit log)
- Leveled compaction: 111 GB writes

Example 1 - note

- Leveled compactions write amplification is not only a problem with 100% write...
- Can have just 10% writes and an amplified write workload so high that
  - Uncached reads slowed down because we need the disk to write
  - Compaction can't keep up, uncompacted sstables pile up, even slower reads
- Leveled compaction is unsuitable for many workloads with a non-negligible amount of writes even if they seem "read mostly"

Can we create a new compaction strategy with

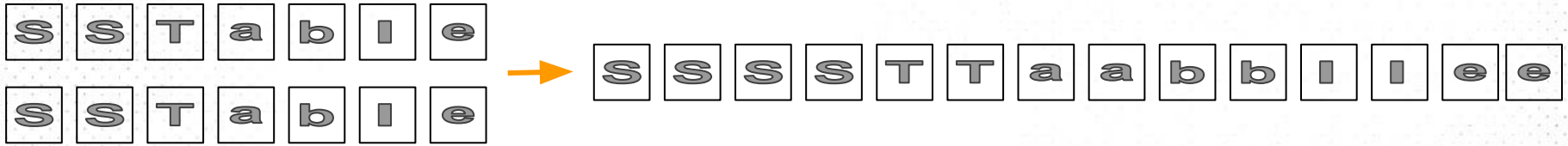
- Low write amplification of size-tiered compaction
- Without its high temporary disk space usage during compaction?

Strategy #3: Hybrid Compaction

- New in upcoming version of **Scylla Enterprise**
- Hybrid of Size-Tiered and Leveled strategies:

Strategy #3: Hybrid Compaction

- Size-tiered compaction needs temporary space because we only remove a huge sstable after we fully compact it.
- Let's split each huge sstable into a **run** (a la LCS) of "fragments":
  - Treat the entire run (not individual sstables) as a file for STCS
  - Remove individual sstables as compacted. Low temporary space.



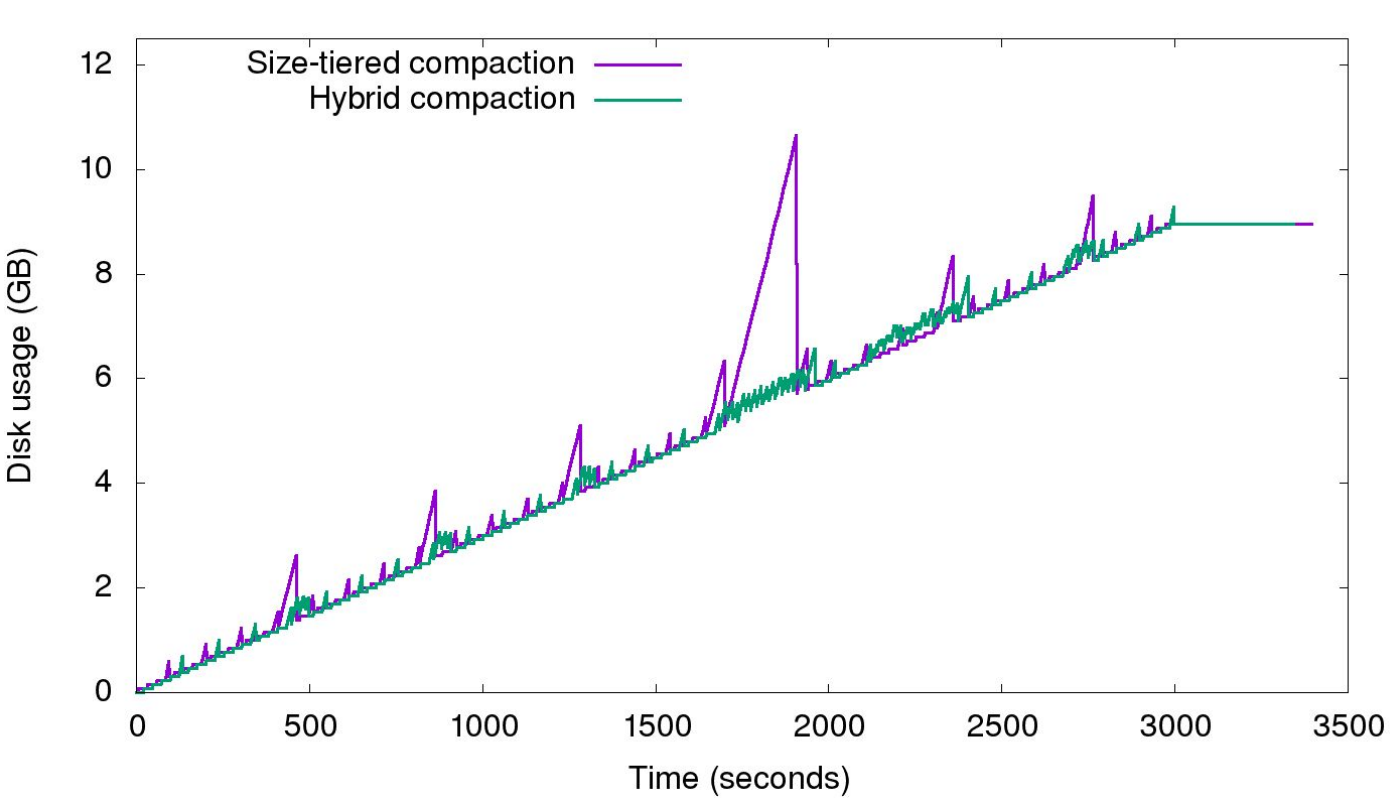
Strategy #3: Hybrid Compaction

- Solve 4x worst-case in overwrite workloads with other techniques:
  - Compact fewer sstables if disk is getting full
    - Not a risk because small temporary disk needs
  - Compact fewer sstables if they have large overlaps

Hybrid compaction - amplification

- **Space amplification**:
  - Small constant temporary space needs, even smaller than LCS (M*S per parallel compaction, e.g., M=4, S=160 MB)
  - Overwrite-mostly still a worst-case, but 2-fold instead of 5-fold
  - Optimal.
- **Write amplification**:
  - O(logN), small constant — same as Size-Tiered compaction
- **Read amplification**:
  - Like Size-Tiered, at worst O(logN) if updating the same partitions

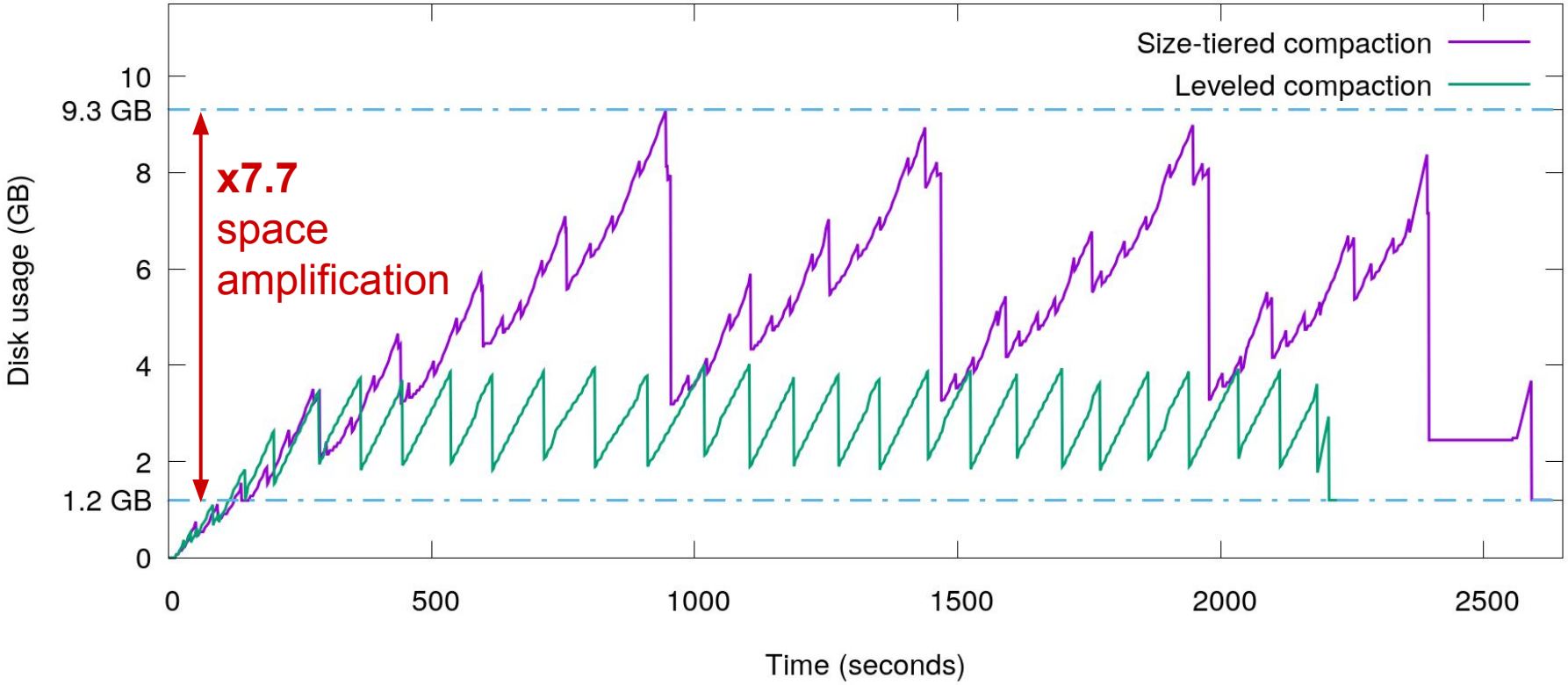
Example 1, with Hybrid compaction strategy



Example 2 - overwrite workload

- Write 15 times the same 4 million partitions
  - cassandra-stress write n=4000000 -pop seq=1..4000000 -schema "replication(strategy=org.apache.cassandra.locator.SimpleStrategy,factor=1)"
  - In this test cassandra-stress not rate limited
  - Again, small (10MB) LCS tables
- Necessary amount of sstable data: 1.2 GB
- STCS space amplification: x7.7 !
- LCS space amplification lower, constant multiple of sstable size
- Hybrid will be around x2

Example 2


Size-tiered compaction
Leveled compaction
10
9.3 GB
8
6
4
2
1.2 GB
0
Disk usage (GB)
x7.7
space
amplification
0
500
1000
1500
2000
2500
Time (seconds)

Example 3 - read+updates workload

- **When workloads are read-mostly, read amplification is important**
- When workloads also have updates to existing partitions
  - With STCS, each partition ends up in multiple sstables
  - Read amplification
- An example to simulate this:
  - Do a write-only update workload
    - cassandra_stress write n=4,000,000 -pop seq=1..1,000,000
  - Now run a read-only workload
    - cassandra_stress read n=1,000,000 -pop seq=1..1,000,000
    - measure avg. number of read bytes per request

Example 3 - read+updates workload

- Size-tiered: **46,915** bytes read per request
  - Optimal after major compaction - 11,979
- Leveled: **11,982**
  - Equal to optimal because in this case all sstables fit in L1...
- Increasing the number of partitions 8-fold:
  - Size-tiered: **29,794** ← *luckier this time*
  - Leveled: **16,713** ← *unlucky (0.5 of data, not 0.9, in L2)*
- BUT: Remember that if we have non-negligable amount of writes, LCS write amplification may slow down reads

Example 3, and major compaction

- We saw that size-tiered **major compaction** reduces read amplification
- It also reduces space amplification (expired/overwritten data)
- Major compaction only makes sense if very few writes
  - But in that case, LCS's write amplification is not a problem!
  - So LCS is recommended instead of major compaction
    - Easier to use
    - No huge operations like major compaction (need to find when to run)
    - No 50%-free-disk worst-case requirement
    - Good read amplification and space amplification

Why major compaction? Is it suboptimal? (from STCS perspective)

- STCS is quite inefficient / slow at getting rid of obsolete data (droppable tombstone, shadowed data).
  - For droppable tombstone, there's tombstone compaction. Suboptimal though.
  - For shadowed (overwritten) data, there's nothing to do. Just wait for data and obsolete data to be compacted together after reaching same tier.

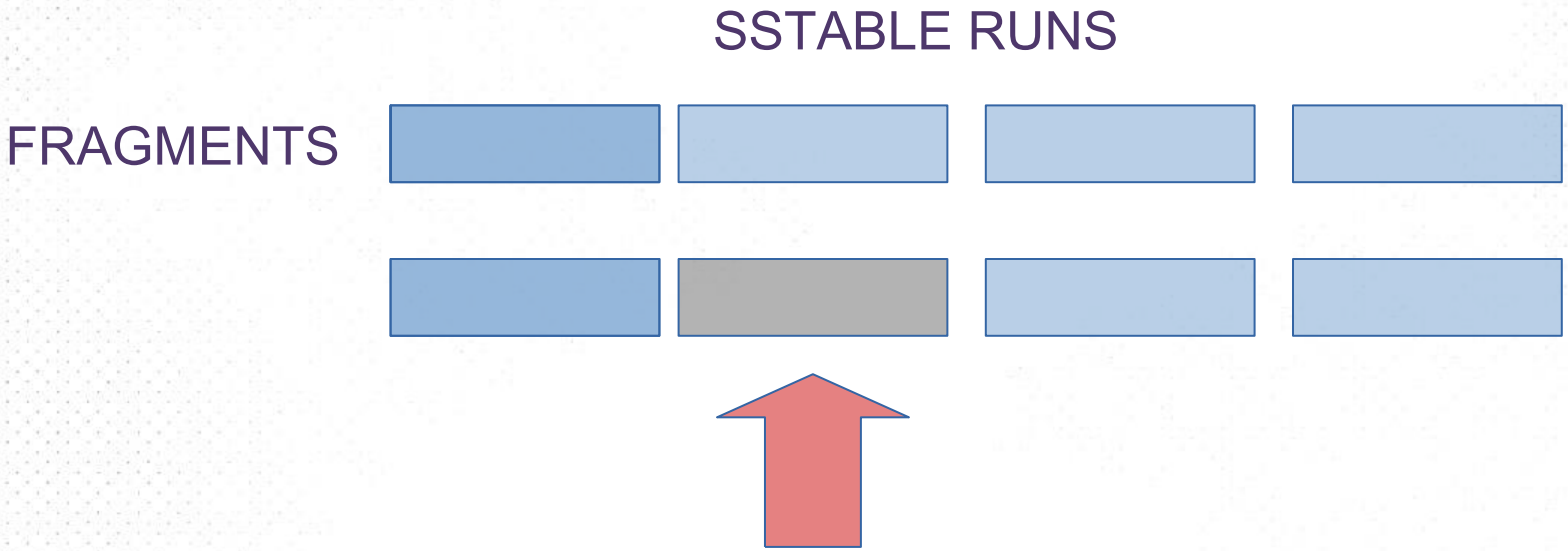
Tombstone compaction

- Triggered when standard compaction has nothing to do
- Tombstone compaction selects sstable with a percentage of droppable tombstone higher than N% and hopes space will be released.
- That's suboptimal though…
- Tombstone cannot be purged unless it's compacted with data it deletes/shadows.
- CASSANDRA-7019 suggests improving the feature by compacting a sstable with older overlapping sstables. That will be inefficient with STCS though. What can we do instead?

Making improved tombstone compaction efficient with hybrid

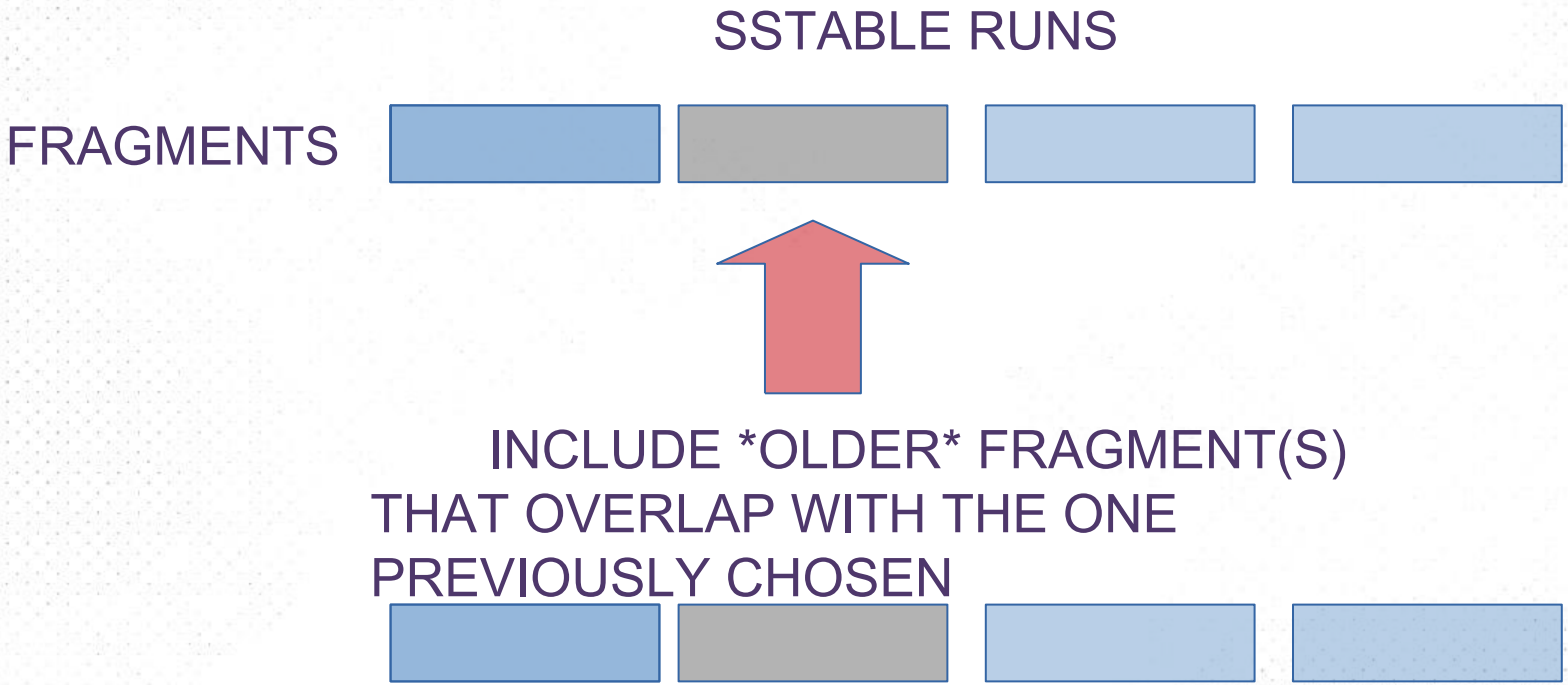
- Hybrid can choose a fragment from high tiers and compact it with all overlapping fragments from sstable runs of same tier or above.
- All sstable run(s) involved will have their (often only one) fragment replaced by another with: (LIVE DATA) – (SHADOWED DATA) – (DROPPABLE TOMBSTONES)
- Temporary space requirement of N * fragment size, N = number of fragments involved
- Make it optional for regular scenarios but use it if running out of disk space.

Hybrid tombstone compaction - Example

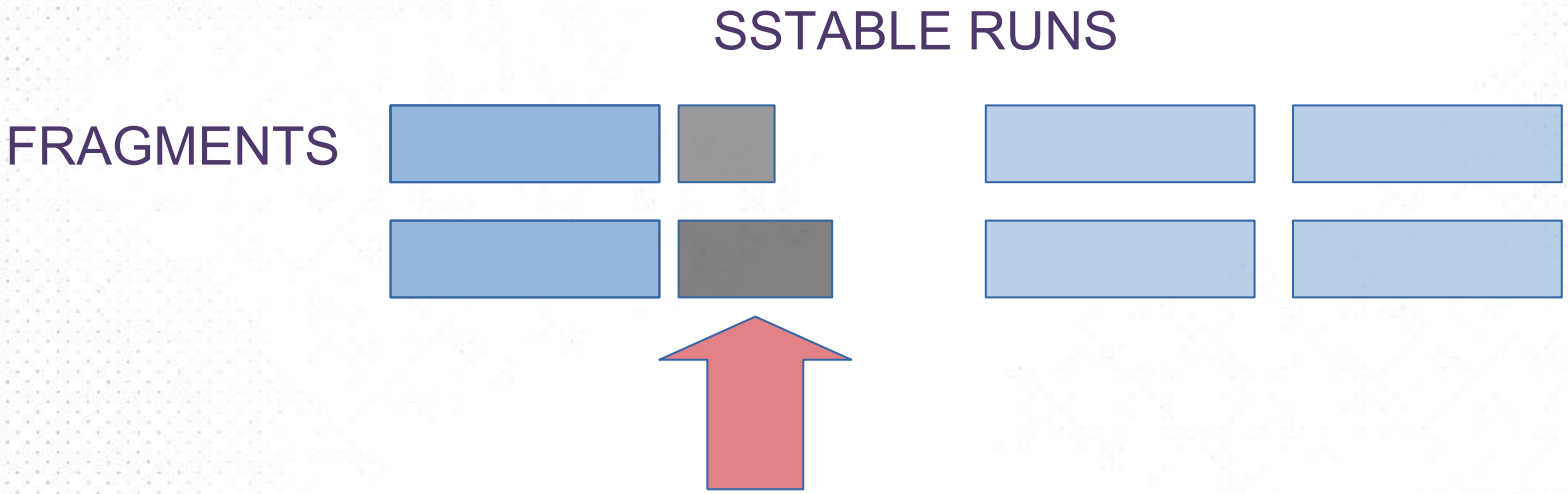


CHOOSE A SSTABLE RUN FRAGMENT WITH N% OF DROPPABLE TOMBSTONES

Hybrid tombstone compaction - Example



Hybrid tombstone compaction - Example



REPLACE FRAGMENTS BY ONES WITHOUT SHADOWED DATA AND DROPPABLE TOMBSTONES

Making hybrid take action when lots of duplicate data waste disk space

- Compact fewer tables of same tier if they contain lots of duplicate data. Affects only overwrite intensive workloads.
- Cardinality information may help us estimating duplication between tables. Work only at partition level though…
- Nadav came up with idea of doing a compaction sample to help with estimation at clustering level. Works due to murmur tokenizer.
- At worst case (running out of space), Hybrid can afford to compact biggest tiers together to get rid of all obsolete data with low temporary space requirement.

Conclusion on this hybrid strategy topic

- Goal is to have hybrid do the cleanup job itself rather than relying on sysadmin to run manual (major compaction) at an interval.
- Hybrid can take smart decisions due to its nature; non-aggressive, incremental steps towards improving space amplification without hurting system performance like major does.
- Trying to bring best of both worlds.

Strategy #4: Time-Window Compaction

- Introduced in Cassandra 3.0.8, designed for **time-series data**
- Replaces *Date-Tiered compaction strategy* of Cassandra 2.1 (which is also supported by Scylla, but not recommended)

Time-Window compaction strategy (cont.)

In a time-series use case:

- Clustering key and write time are correlated
- Data is added in time order. Only few out-of-order writes, typically rearranged by just a few seconds
- Data is only deleted through expiration (TTL) or by deleting an entire partition, usually the same TTL on all the data
- The rate at which data is written is nearly constant
- A query is a clustering-key range query on a given partition
  Most common query: "values from the last hour/day/week"

Time-Window compaction strategy (cont.)

- Scylla remembers in memory the minimum and maximum clustering key in each newly-flushed sstable
  - Efficiently find only the sstables with data relevant to a query
- Other compaction strategies
  - Destroy this feature by merging “old” and “new” sstables
  - Move all rows of a partition to the same sstable…
    - But time series queries don’t need all rows of a partition, just rows in a given time range
    - Makes it impossible to expire old sstable’s when everything in them has expired
    - Read and write amplification (needless compactions)

Time-Window compaction strategy (cont.)

So TWCS:

- Divides time into “time windows”
  - E.g., if typical query asks for 1 day of data, choose a time window of 1 day
- Divide sstables into time buckets, according to time window
- Compact using Size-Tiered strategy inside each time bucket
  - If the 2-day old window has just one big sstable and a repair creates an additional tiny “old” sstable, the two will not get compacted
  - A tradeoff: slows read but avoids the write amplification problem of DTCS
- When time bucket exits the current window, do a major compaction
  - Except for small repair-produced sstables, we get 1 sstable per time window

Summary

Workload	Size-Tiered	Leveled	Hybrid	Time-Window
Write-only	2x peak space	2x writes	**Best**	-
Overwrite	Huge peak space	write amplification	**high peak space, but not like size-tiered**	-
Read-mostly, few updates	read amplification	**Best**	read amplification	-
Read-mostly, but a lot of updates	read and space amplification	write amplification may overwhelm	**read amplification**	-
Time series	write, read, and space ampl.	write and space amplification	write and read amplification	**Best**

THANK YOU

Any questions?

Please stay in touch

nyh@scylladb.com