Compaction Enhancements: Increased Storage Density and Time Series Made Much Easier

Raphael S. Carvalho
Software Engineer

SCYLLA

Raphael S. Carvalho

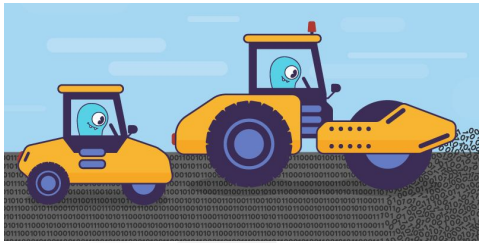
Software Engineer at ScyllaDB

- Member of the ScyllaDB storage team
- Responsible for the compaction subsystem
- Previously worked on Syslinux and OSv

SCYLLA

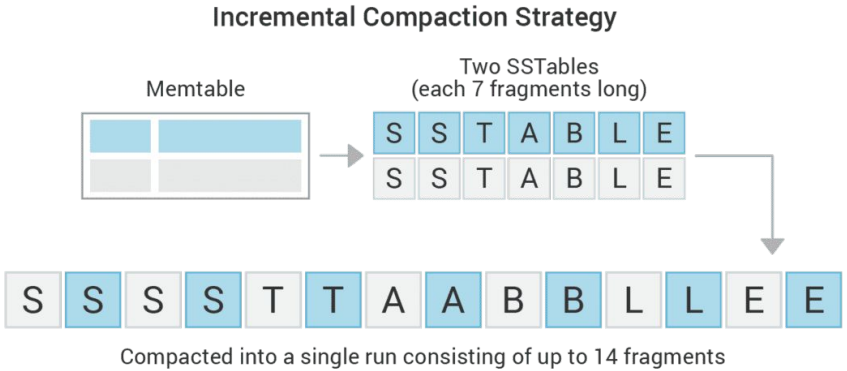
Agenda

- Space optimization for incremental compaction
- “Bucketless” time series, i.e. time series made much easier for you
- Upcoming improvements



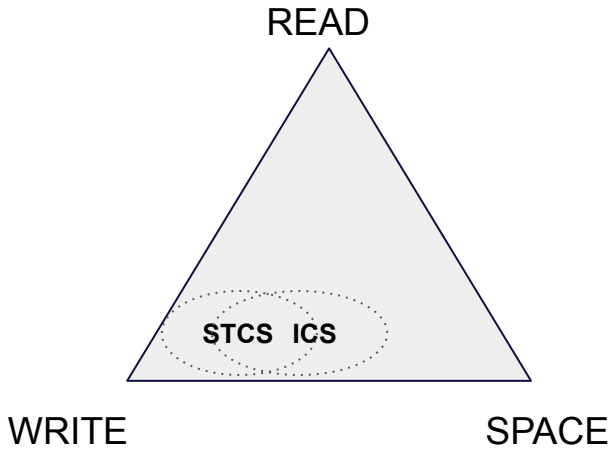
Let's take a look back

- **Incremental compaction (ICS)** introduced in enterprise release 2019.1.4
- Known for combining techniques from both size-tiered and leveled strategies
- Fixes the 100% space overhead problem in size-tiered compaction, increasing disk utilization significantly.



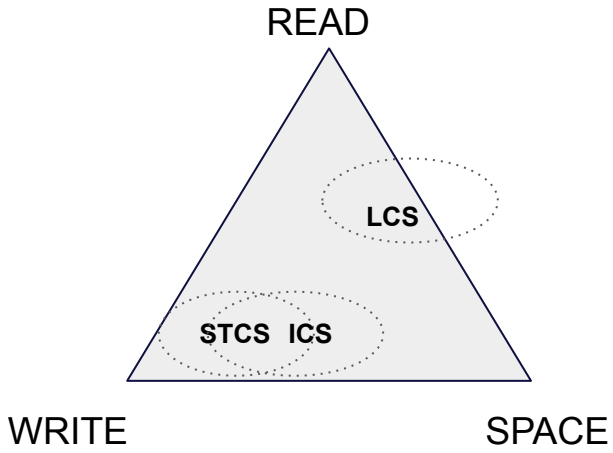
Is it enough?

- Space overhead in tiered compaction was efficiently fixed, however…
- Incremental (ICS) and size-tiered (STCS) strategies share the same space amplification (~2-4x) with overwrite workloads, where:
  - They cover a similar region in the three-dimensional efficiency space, also known as RUM conjecture trade-offs.



Turns out it's not enough. But can we do better?

- Leveled strategy and Size-tiered (or ICS) cover different regions
  - Interesting regions cannot be reached with either strategies.
  - But interesting regions can be reached by combining data layout of both strategies
    - i.e. a hybrid (tiered + leveled) approach



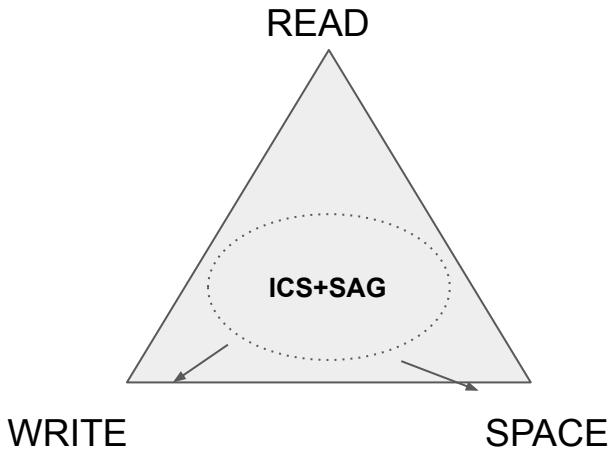
Let’s work to optimize space efficiency then

- A few high-level goals:
  - Optimize space efficiency with overwrite workloads
  - Ensure write and read latency meet SLA requirements

- That’s Space Amplification Goal (SAG) for you.
- Increased storage density per node? YES.
- Reduce costs? YES.

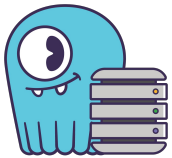
A few facts about this feature

- This feature (available since Scylla Enterprise 2020.1.6) can only be used with Incremental Compaction
- Compaction will dynamically adapt to the workload to meet requirements
- Under heavy write load, compaction strategy will work to meet write latency requirement.
- Otherwise, strategy works to optimize space efficiency to the desired extent
- Translates into:
  - Storage Density per node ++
  - Costs --
  - Scale ++



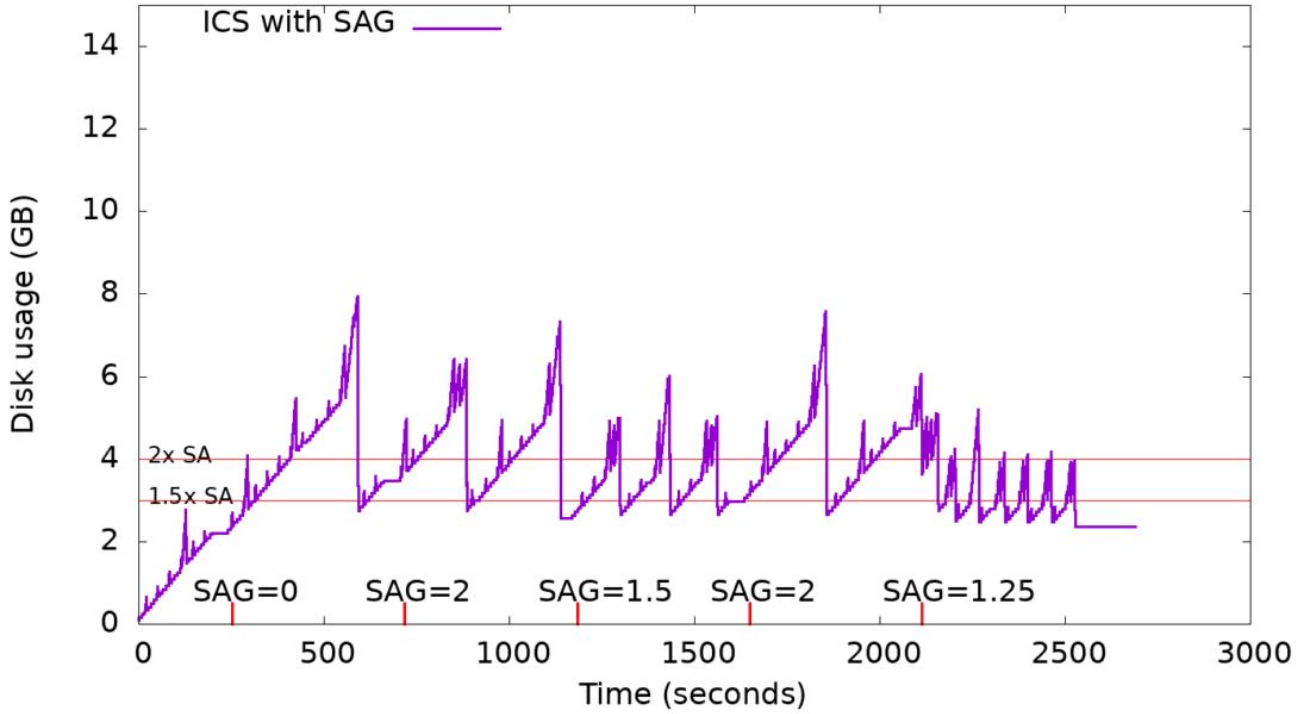
Enabling the space optimization (SAG)

```
ALTER TABLE keyspace.table
   WITH compaction = {
      'class': 'IncrementalCompactionStrategy',
      'space_amplification_goal': '1.5',
   };
```



- This will enable the feature with a space amplification goal of 1.5
- The lower the configured value the higher the write amplification
- Adaptive approach minimizes the impact of extra amplification
- Gives user control to reach interesting regions in the three-dimensional efficiency space

Space optimization in action…



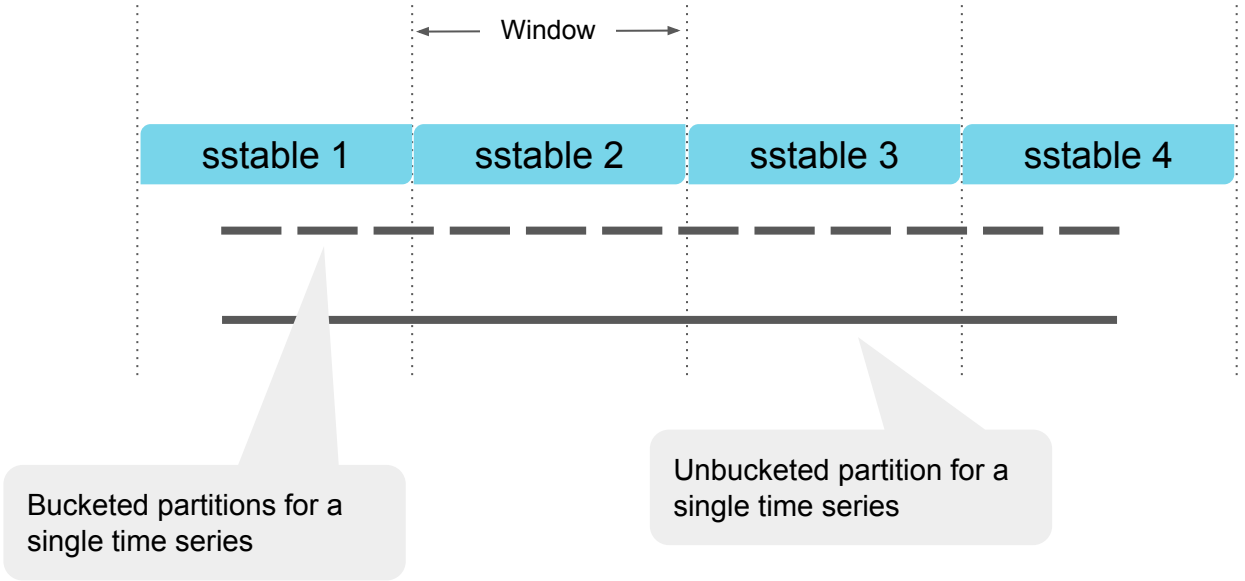
A common schema for time series looked like…

```
CREATE TABLE billy.readings (
    sensor_id int,
    date date,
    time time,
    temperature int,
    PRIMARY KEY ((sensor_id, date), time)
)
```

Why bucket in time series?

- Large partitions were known to causing all sorts of problems
  - Index inefficiency when reading from the middle of a large partition
  - Latency issues when repairing large partitions
  - High resource usage and read amplification when querying multiple time windows
  - Reactor stalls which caused higher P99 latencies
  - And so on…
- Consequently applications were forced to “bucket” partitions to keep their size within a limit.

Bucketed vs Unbucketed time series



Time series made much easier for you!

- But those bad days are gone!
  - Scylla allows a large partition to be efficiently indexed: O(logN)
  - Scylla's row-level repair allows large partitions to be efficiently repaired
  - TimeWindowCompactionStrategy can now efficiently query multiple time windows
    - by discarding SSTable files which time range is irrelevant for the query
    - Incrementally open the relevant files to reduce resource overhead
    - Therefore, read amplification and resource usage problems are fixed

A schema for time series can now look like…

```
CREATE TABLE billy.readings (
    sensor_id int,
    time time,
    temperature int,
    PRIMARY KEY (sensor_id, time)
)
```

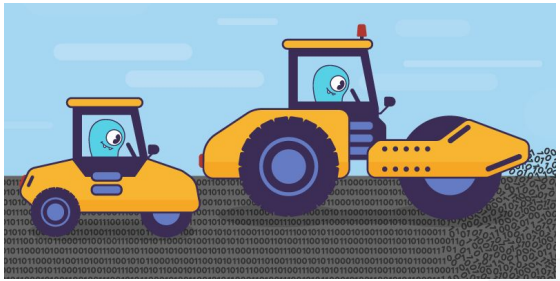
- There's no longer any field date in schema, meaning that:
  - Application won't have to create new partitions on a fixed interval for a time series
  - Querying a time series will be much easier as only a single partition is involved
- Bucketing days are potentially gone!
- Lots of complexity reduced in the application side

Upcoming improvements

- Compaction becoming overall more resilient / performant:
  - Changes were recently made to make Cleanup, Major compactions more resilient when system is running out of disk space
  - Dynamic control of compaction fan-in to increase overall compaction efficiency
    - Based on observation that efficiency is a function of number of input files and their relative sizes
    - Don't dilute the overall efficiency by submitting jobs which efficiency is greater than or equal to efficiency of ongoing jobs
    - Tests show that write amplification is reduced under heavy write load while keeping space and read amplifications within bounds
    - Makes the system adapt even better to changing workloads
    - More stability. More performance.

Upcoming improvements

- Reduce compaction aggressiveness by:
  - Improvements in I/O scheduler (Pavel Emelyanov covers this in depth in his talk)
  - Improvements in Compaction backlog controller
  - Aiming at improving tail latency and overall system stability.
- Off-strategy compaction (Asias He covers this better)
  - Make repair-based node operations more efficient, faster
  - Consequently, better elasticity



Thank you!

Stay in touch

Raphael S. Carvalho

@raphael_scarv

raphaelsc@scylladb.com