This doc is shared publicly

Support for out-of-order samples in the TSDB

- Authors: Ganesh Vernekar, Jesús Vázquez, Dieter Plaetinck
- 2022-02-01: Started work on draft.
- 2022-04-08: First shared version.
- 2022-04-13: Shared publicly with the Prometheus Community.
- 2022-08-01: Submitted pull request in Prometheus with the implementation

Publicly shared

Introduction

Problem statement

Prometheus TSDB has two important restrictions regarding the age of the samples it's ingesting:

- Samples can't be older than the newest one for the same series
  - Let's call this *ingesting OOO samples* for the rest of the doc
- Samples can't be more than 1h older than the newest sample among all series
  - (assuming default 2h blocks configuration)
  - Let's call this *ingesting old samples* for the rest of the doc

Why is this an issue?

- Metric producers isolated from a network connection for more than one hour trying to push *old samples* which will be rejected and metrics would be lost forever, if samples (of other producers) were still ingested successfully.
- Complex metric delivery architectures using message buses like kafka might have delays due to congestion, randomized sharding, or preprocessing resulting in samples lost.
- Some metric producers continuously aggregate data for the past few hours and then send the aggregation result for an old timestamp. This aggregation may receive updates over time.
- IOT devices that save battery by disconnecting from the network for long periods of time can't reliably push metrics.

This doc is shared publicly

Overview

At a high level, we have ingestion (part 1), querying (part 2), durability of ingestion (part 3), and background maintenance via compaction (part 4) that we discuss in this design doc for out-of-order samples.

For this design, ingesting OOO samples and ingesting old samples, as defined in the problem statement, will be supported out of the box and are not seen as a different thing. It does not matter how old the sample is - it can take it all. But to protect the TSDB from receiving way too old samples and the unexpected scenarios that may lead to, like compacting samples into too many 2 hour blocks, there will be a configuration to limit how old a sample can be that is being ingested.

> The gray colored boxes (this one for example) starting with the title “Excursus” contain some additional low level details of how we are implementing/designing some parts of the implementation. If you are here to consume the doc quickly and at a not-so-low level, feel free to skip those boxes.

There is a Q&A section at the end of this document.

This design is actively being worked on in the [out-of-order branch of grafana/mimir-prometheus repo].

Non goals for the initial version

- Deletions: They are not supported yet. Temporary solution is to wait for out-of-order data to be compacted into a persistent block, and then use the normal deletion that we have today (so technically it allows a delayed deletion of out-of-order data)
- No isolation for the out-of-order samples: If you started a query that would touch the out-of-order data, and if there is an out-of-order sample ingested in parallel, that sample may or may not be included in the query result.

Glossary and Background

Feel free to come to this later if you get confused while reading the rest of this document.

- TSDB: Time Series Database. Specifically talking about the Prometheus TSDB.
- Head block: The part of TSDB that stores the data in the memory. See the diagrams and explanation here if you want more information.
- Chunk: A collection of samples. Might or might not be compressed.
- Head chunk / active chunk: A chunk that is still in the memory and has not been flushed to the disk.
- Compaction: Background maintenance done in TSDB to merge one of more persistent blocks and convert head block into a persistent block.

- Persistent block: Part of TSDB which contains immutable data. See this for visual representation, and this for more details.
- M-map files / M-mapped / Memory-mapped: The data from the Head block are frequently flushed to disk and memory-mapped (m-mapped). More about m-mapped file here. More about m-mapping in TSDB here.
- WAL / Write-Ahead-Log. Used by TSDB to give durability prior to writing samples. Wikipedia definition. More details on Prometheus WAL is here.
- WBL / Write-Behind-Log: Similar to WAL. But the logs are written to disk *after* writing samples.

Part 1: Ingestion of out-of-order samples and memory mapping

While developing a solution for out-of-order support, one of the main challenges is to avoid duplicating the series definition and the index that TSDB Head stores in memory: those use a lot of memory (Grafana Labs has data showing up to 20% of memory consumption for scraped labels in some of our clusters), so we just want to reuse the same in-memory series definition that currently stores the active chunk and a slice of memory-mapped chunks references and expand it with a new active chunk to only store the out-of-order samples (let's call it *oooHeadChunk*) and another slice of memory-mapped chunk references for the out-of-order chunks. These additional chunks do not take any space upfront, other than their 8 byte pointer references, and are initialized only when an out-of-order sample comes in. So the memory overhead will be mostly proportional to the number of series getting out-of-order samples.

Today's TSDB drops incoming out-of-order samples on arrival; they're rejected instead of being written into the head chunk. This is the main change: we ingest the out-of-order samples into the above-mentioned oooHeadChunk, **which will store sorted uncompressed samples for the series**. The sorted nature of this chunk allows us to cheaply query the sample as soon as it has been inserted. **oooHeadChunk allows inserting samples at any position**.

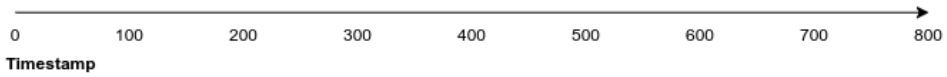
The out-of-order chunks once "full" (i.e. 30 samples by default to use less memory, but this will be configurable) will be converted to a XOR encoded chunk, the common format currently used by the TSDB, and flushed onto the disk and memory mapped like we do already in TSDB. **We will share the memory mapped files for in-order and out-of-order chunks**, hence no additional artifacts here. We will add one bit to the memory mapped chunk metadata to identify the out-of-order chunks.

Example of ingestion

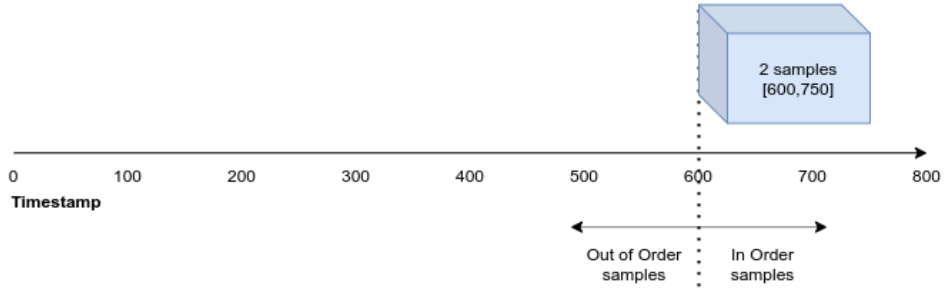
Let's take a look at how the TSDB will handle incoming out-of-order samples with these changes in the ingestion path.

This doc is shared publicly

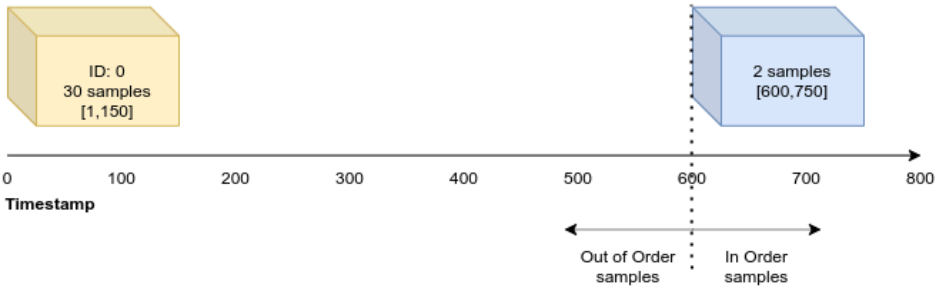
1) initial state: empty series



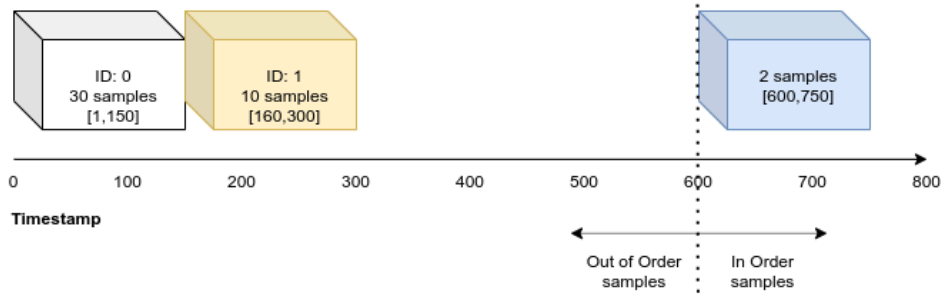
2) 2 samples arrive, one with ts 600 and one with ts 750



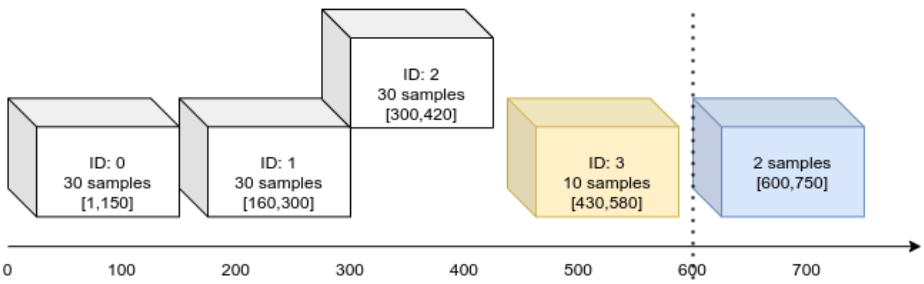
3) 30 samples arrive with ts between 1 and 150
they are out of order with respect to samples already present



4) 10 more samples arrive, continuing the same sequence. The previous chunk was full and is mmapped. a new head chunk for OOO data is created



5) more samples arrive, creating more chunks. Note here both chunk 1 and 2 have a value for ts 300, causing overlap



This doc is shared publicly

This demonstrates the nature of out-of-order samples. The latest chunk could still have the oldest sample that was emitted till now. An in-order chunk would never accept a sample that would make it overlap with the m-map chunks. Memory mapped chunks are immutable, so appending the sample to a specific chunk is not a viable option.

The benefits:

- Samples are queryable from arrival which follows prometheus guarantee of read after write.

The downsides:

- Cost of OOO samples in memory is fairly high. We think we can contain the cost by keeping small chunks of 30 samples. At Grafana Cloud our stats show that OOO samples are far less frequent than in-order samples, about 1% of the total.
- When samples are ingested as OOO and conflict (same TS, different value) with previously seen samples that went into a different chunk, we cannot increment the right ingest error metrics, and the guarantee of this proposal is that we will keep only one of conflicting samples in the resulting blocks after compaction.

Excursus: The new Chunk implementation that will contain the uncompressed sorted samples

Currently the TSDB uses a Chunk implementation called XORChunk which stores compressed samples in the memory. OOOChunk is a simple slice of uncompressed samples which allows inserting samples at any position, hence keeping the slice sorted w.r.t. the timestamp of the samples.

OOOChunk gets converted into a XOR encoded chunk eventually when the chunk is flushed onto the disk by m-mapping it.

Part 2: Querying out-of-order samples

With the insights gained in Part 1 we now know that there may be two active chunks for each series in the memory, for in-order and out-of-order data respectively. So when a query comes in, we need to be able to read the samples from either or even both of them depending on the query interval. We also need to support including in-order and out-of-order mmapped chunks. This is covered below. Chunks that have been compacted in blocks need no further specific attention as we simply rely on Prometheus' vertical compaction support.

This doc is shared publicly

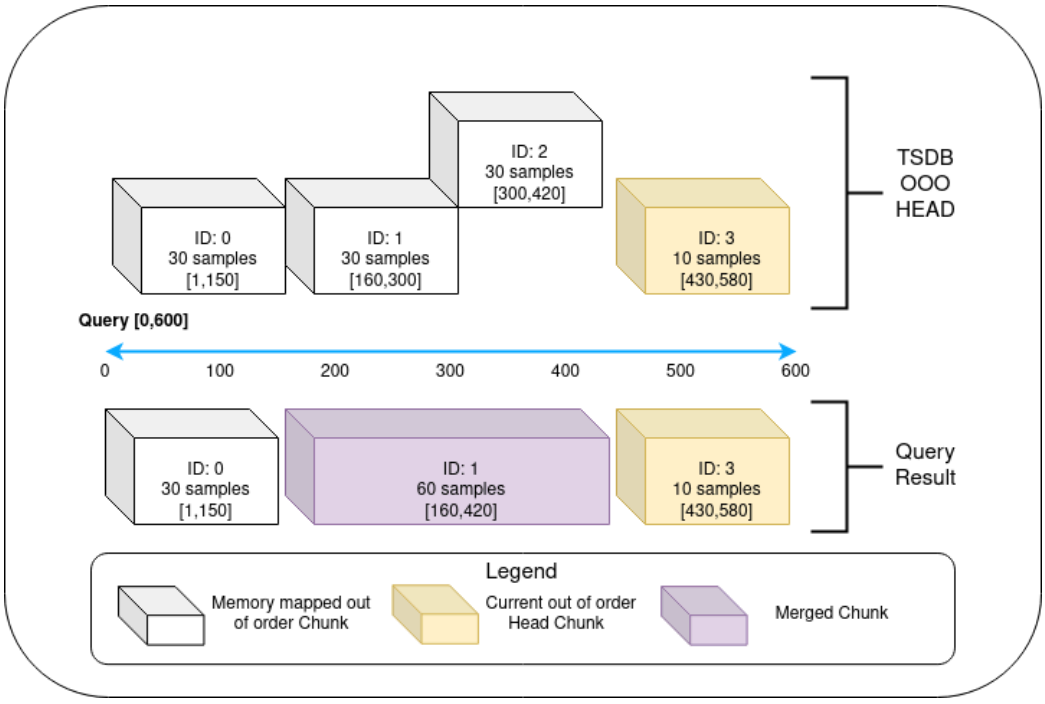
Reading chunks from the out-of-order head

In the ingestion section above we made the tradeoff of consuming a bit more memory but ensuring that samples were sorted and queryable so that the querying did not become an expensive operation and to maintain today's guarantee of read after write.

That said there is still one complexity we need to solve, which is that given the nature of out-of-order samples, the out-of-order chunks:

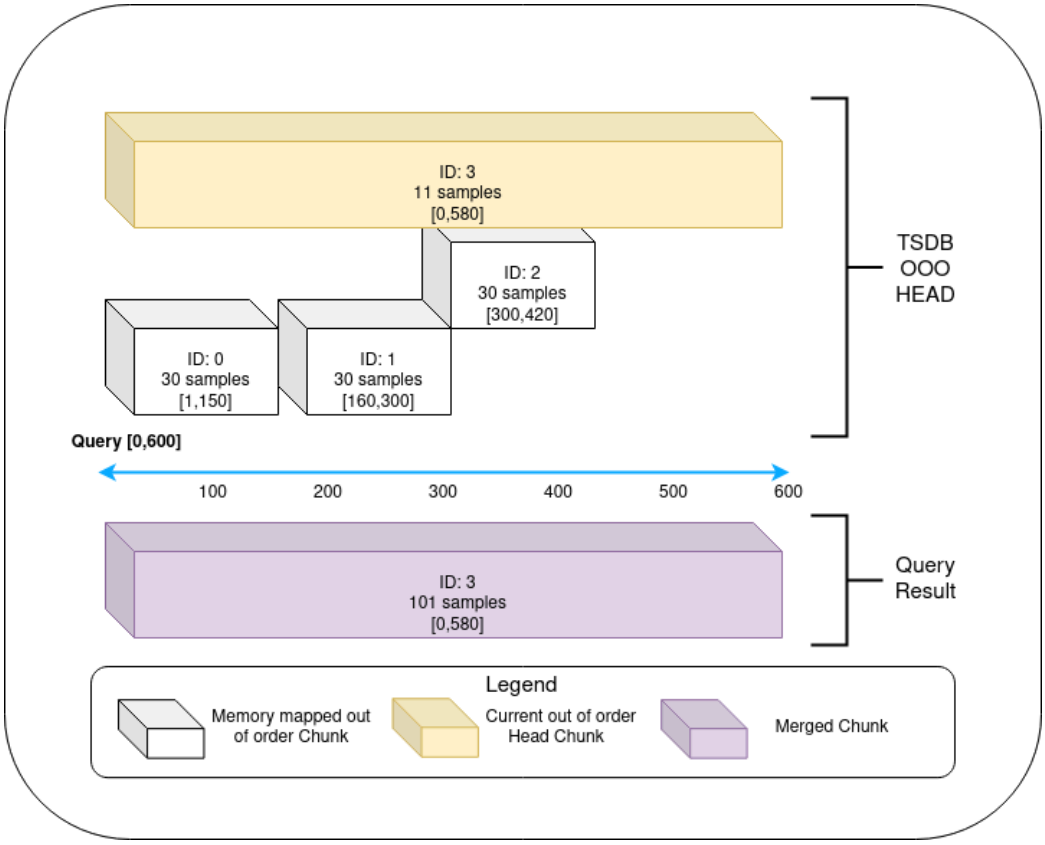
- Could overlap each other, for example: we could receive 30 out of order samples between timestamps [100,200] followed by another 30 between timestamps [150,200].
- Be in a non-trivial order, for example: we could receive 30 out of order samples between timestamps [300,400] followed by another 30 samples between [100,200].

For this reason every time a query comes in we need to inspect all the chunks and calculate which ones have samples for the given query interval and once we find them we need to merge them into a single chunk. This merged chunk is a new concept and we need to clarify that it is not stored, it is only the response the TSDB gives to the upper layer for the query response.



In the above image, two of the chunks are overlapping on a single sample.

This doc is shared publicly



In the above image all chunks are overlapping (1 and 2 at timestamp 300 and then all of them with Chunk 3). The head will return a chunk with the ID of the chunk that has the first sample of the query interval, in this case the sample at timestamp 0. Then when the chunk is iterated it will go through all the samples.

Finding which chunks overlap and composing the merged chunk is not an expensive operation since most of what is happening is looking at the chunks metadata and then creating a slice of chunks that need to be merged. The expensive part which is reading the samples will only be called once and the current implementation guarantees that the result will be ordered by timestamp.

Some guarantees given by the query:

- Once a query has started, let's say the active out-of-order chunk was ID:3, if more samples come in parallel and end up creating a chunk ID:4, the new chunks will be invisible to the query and only upto the chunk ID:3 will be considered.
- Similar to the above, once a query has started, if the chunk ID:3 had samples in the range [t1, t2], if it gets a sample in parallel that is outside this range, that is also invisible to the query. But if the sample happens to be within [t1, t2], it may or may not appear in the query,
- If there are multiple samples with the same timestamp because of overlapping chunks, the query will only see one sample among the conflicting samples.

This doc is shared publicly

How will TSDB combine the out-of-order samples with in-order data and persistent blocks with head blocks.

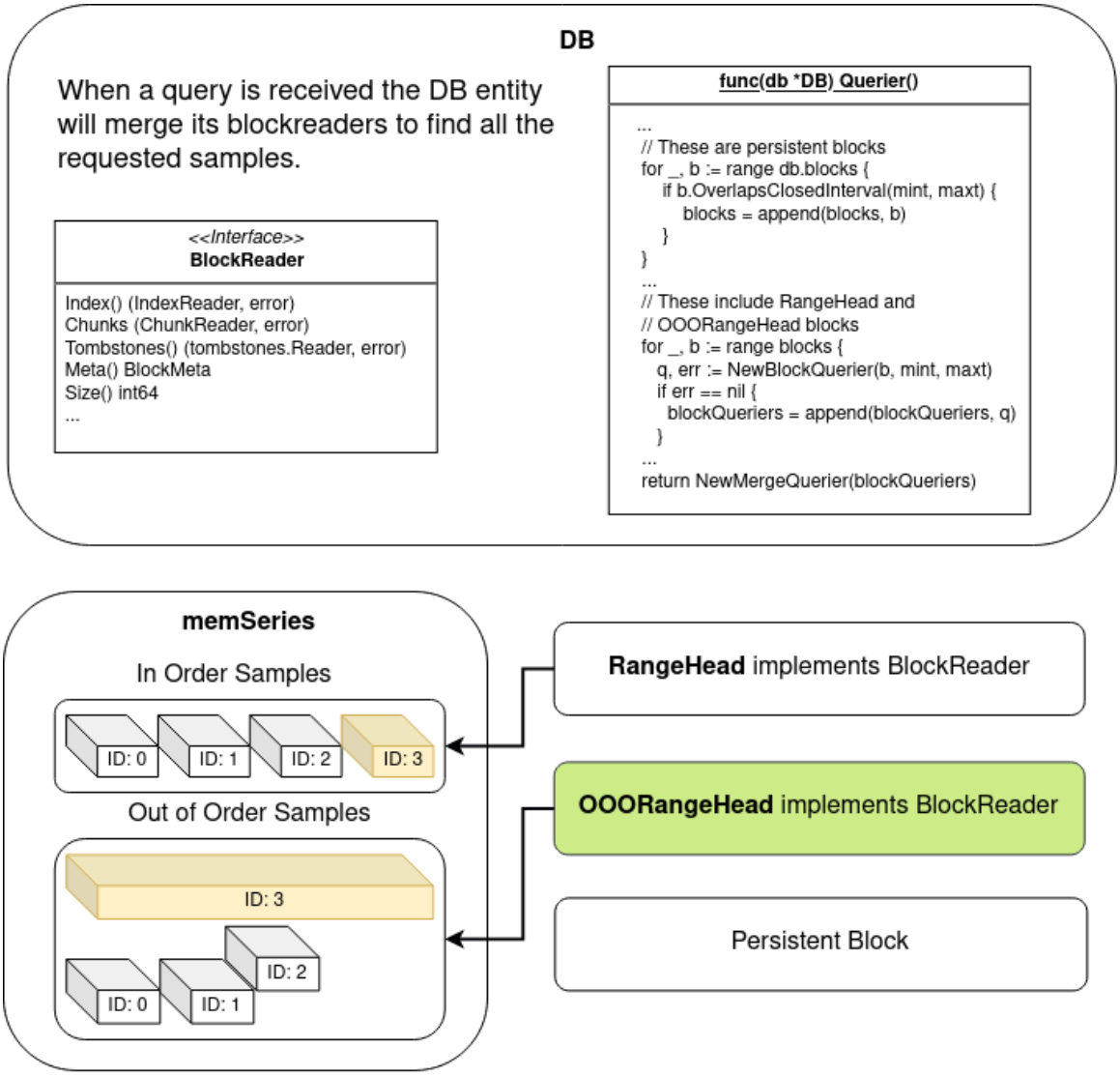
Today the TSDB uses an abstract BlockReader interface which is used to read data from individual persistent blocks and the in-memory head block. The existing RangeHead implements the BlockReader interface to read the in-order data from the head block.

Taking inspiration from that, we create another wrapper for the head block OOORangeHead that only provides access to the out-of-order data. These two wrappers are not only used for querying but also for compacting and that is something that will be discussed in Part 4 of this document.

This existing design of the TSDB is what would in theory make the out-of-order support a non-breaking change. The OOORangeHead acts like yet another logical block that provides non-overlapping out-of-order chunks by abstracting away the above merging of overlapping chunks under it. This logical block may or may not overlap with other blocks in the query, which is already being taken care of by the TSDB via vertical compaction.

Let’s bring up a diagram that shows how the BlockReader abstraction works.

This doc is shared publicly



The benefits:

- There are abstractions in place in TSDB that we can reuse to make this happen without too many breaking changes.

Part 3: Separate Write-”Behind”-Log (WBL) for out-of-order samples and replay of data on startup

Why write-“behind”-log and not write-ahead-log? We can tell if a sample was added to the TSDB as an in-order or an out-of-order sample only after the sample is added (i.e. commit the samples) to the TSDB, because any append in the parallel could make a sample out-of-order while being inferred as an in-order sample initially. Hence write-*behind*-log since we append out-of-order samples logs after it has been added to the TSDB.

Why a separate log? When we truncate WAL currently for cleanups, we retain or discard samples based on their timestamp. If they are after a certain timestamp, we keep it, else discard it. This won't work with out-of-order samples since by definition they will be older than the current normal functioning TSDB, and we cannot efficiently identify the out-of-order samples in the WAL that have been compacted into a block based on their timestamp. Having a separate WBL allows us to manage cleanup of WBL files based on the files present during the compaction of the block (we compact all the out-of-order data from the Head during compaction, more on that below) and we don't need to look at individual samples.

The WBL gets only the out-of-order samples, while the series records go into the old WAL as usual. During WAL replay, the old WAL will be replayed completely first (hence recreating the memory series from the series records), and then WBL will be replayed into the out-of-order section of the TSDB.

Excursus: "m-map markers" in the WBL

Currently, when we look at a sample in the WAL, we can tell if it has been already covered by the m-map chunks if the sample's timestamp is less than the max time of the last m-map chunk. We cannot do that for out-of-order samples since the sample might or might not be covered by the m-map chunk while still overlapping with it.

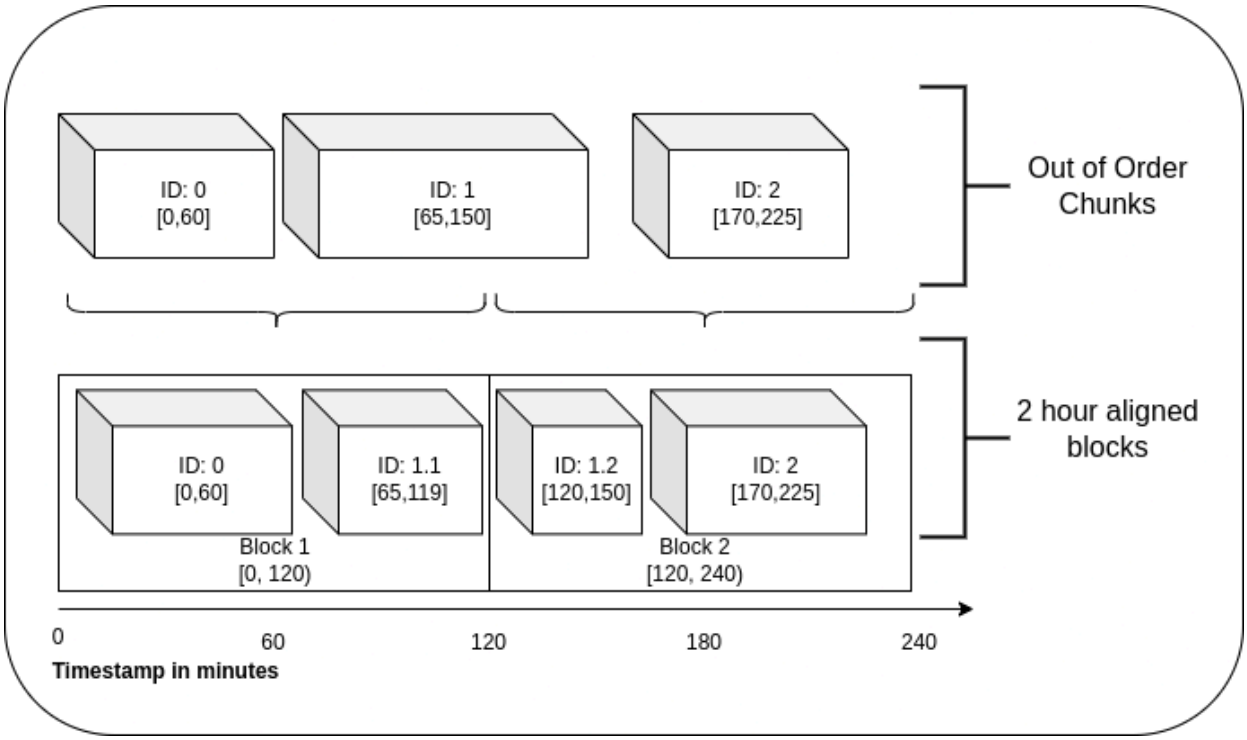
If we cannot say if a sample has been m-mapped, we will end up creating duplicate m-map chunks during the WBL replay. To avoid it, we add something called "m-map markers" per series as soon as the in-memory out-of-order chunk get's m-mapped, signaling that all samples for the series until now in the WBL have been m-mapped. We use these markers to avoid creating duplicate m-map chunks.

Part 4: Handling compactions of out-of-order chunks

Since the out-of-order samples can span a wide range of time (can be longer than the head block and overlapping multiple blocks), the compaction of out-of-order samples from the Head can produce more than one block. To not create blocks that are overlapping with >1 block on disk, we will create 2h aligned blocks.

For example, if the out-of-order samples span a time range [1h, 5h), it will create 3 blocks with ranges [1h, 2h), [2h, 4h), [4h, 5h). This makes sure that a block produced won't end up making the existing blocks on the disk transitively overlap with each other through this new block. If there are no out-of-order samples for any of the potential time ranges, that block will not be produced.

This doc is shared publicly



This compaction will be done separately from the current Head compaction, and will be done right after we compact the Head for the in-order samples. To keep the code simpler, for out-of-order samples, we will only compact the m-mapped out-of-order chunks so that we don't have to deal with live out-of-order chunks. And to avoid stale series hanging around, we will m-map all the live out-of-order chunks right before compaction.

WBL will be cleaned up to the file that was present right before the compaction.

Excursus: WBL cleanup

Let's say the WBL contains the files numbered 21,22,23,24. When compaction for out-of-order samples is about to start, we cut a new file 25, and record that number for this compaction. Only then we start m-mapping the in-memory chunks for compaction. When compaction is finished, we lookup this number and hence delete all the files that were before 25, i.e. 21,22,23,24 here.

Why does this work? Since we are m-mapping live chunks after cutting a new WBL file, it is assured that all the samples before that new file are present in the m-map chunks and hence have been compacted away into blocks. Hence it is safe to delete those WBL files on a successful compaction.

Won't the new WBL file get samples that go into compaction since we m-map chunks after that, hence creating duplication during replay? Yes, it can, but it is a minor issue. It will end up with another block with duplicate samples, but the vertical compaction blocks will deduplicate the samples eventually.

This doc is shared publicly

Excursus: Handling more out-of-order data while compaction is going on

We will pre calculate the expected time ranges of the resultant blocks after all in-memory chunks have been m-mapped. But once the compaction starts after that, there could be a large stream of out-of-order samples that come that could create more m-map chunks, hence create multiple issues (example, invalidates our earlier calculations of expected time ranges, WBL truncation becomes non-deterministic, querying for out-of-order data for compaction gets more complicated).

Solution to this is taken care of by how querying works. Out-of-order query, once started, would not look beyond the last chunk that was present during the query.

Q&A

(from the comments)

- After chunk ID: 0 [0, 150] is finished, what happens with additional samples with timestamp 70?
    - A: The new chunk gets the sample 70 in that case. Chunks can overlap in the Head. You can see one example in the part 1 above where the active chunk suddenly gets an old sample which makes that chunk to overlap with the other out of order chunks in the Head block.
- What if we are receiving samples "backwards" (150, 149, 148, 147, in this order)
    - A: The active out-of-order chunk (the oooHeadChunk) keeps the samples sorted. It is just a slice of samples. So samples can come in any order and we will keep it sorted in the same chunk and do not have much overhead.
- What happens with samples that are received out of order but overlap with in-order chunks?
    - A: they both are isolated in the read path. The read path will see in-order chunks and out-of-order chunks as different blocks. The merge of those overlaps happens at upper layers where DB merges multiple blocks during the query.
- What happens for series which don't have "in-order" chunks, only "OOO chunks". Are they tracked properly in the head properly?
    - A: This is possible if the series gets a very old sample that happens to be out of bounds (i.e. more than 1hr old). This counts as an active series and everything

    - about the series remains the same in the Head's index, just that it only holds out-of-order samples.
- How do series with only OOO samples and "active series" work together?
    - A: The head block's index does not differentiate between series having in-order samples or only out-of-order samples. It is counted as an active series and the out-of-order samples are flushed into a block at 2h intervals similar to the in-order samples.
- Does WBL replace WAL
    - A: No. It is an additional thing. WAL only holds in-order samples. WBL only holds out-of-order samples and stuff related to the out-of-order samples (like the m-map markers).
- Is there any way to flush in-memory OOO chunks and force compaction?
    - The compaction process includes flushing in-memory chunks as m-mapped chunks as the first step. So a force compaction or not, the in-memory will be flushed to the disk before compaction.
- Can there be ONLY OOO Chunk active?
    - A: Yes. It's possible if a series got a very old sample (more than 1h old) which the TSDB would consider as out of bounds hence does not ingest in the in-order chunks. So we can put it in the out-of-order chunk. The specifics around this are TBD. We can take the maxt of the Head block as the reference to decide if the sample can be ingested in the out-of-order chunk (by looking at the config that decides how old the sample can be).