

ClickHouse memory usage introspection

Nikolai Kochetov



Memory Accounting is complicated

- Allocators cache memory internally
- Memory fragmentation
- Caches
- External libraries
- Direct memory mapping
- MADV_FREE pages
- System metrics are updated with delay
- cgroups and CAP_SYS_ADMIN
- Bugs



Memory leak?

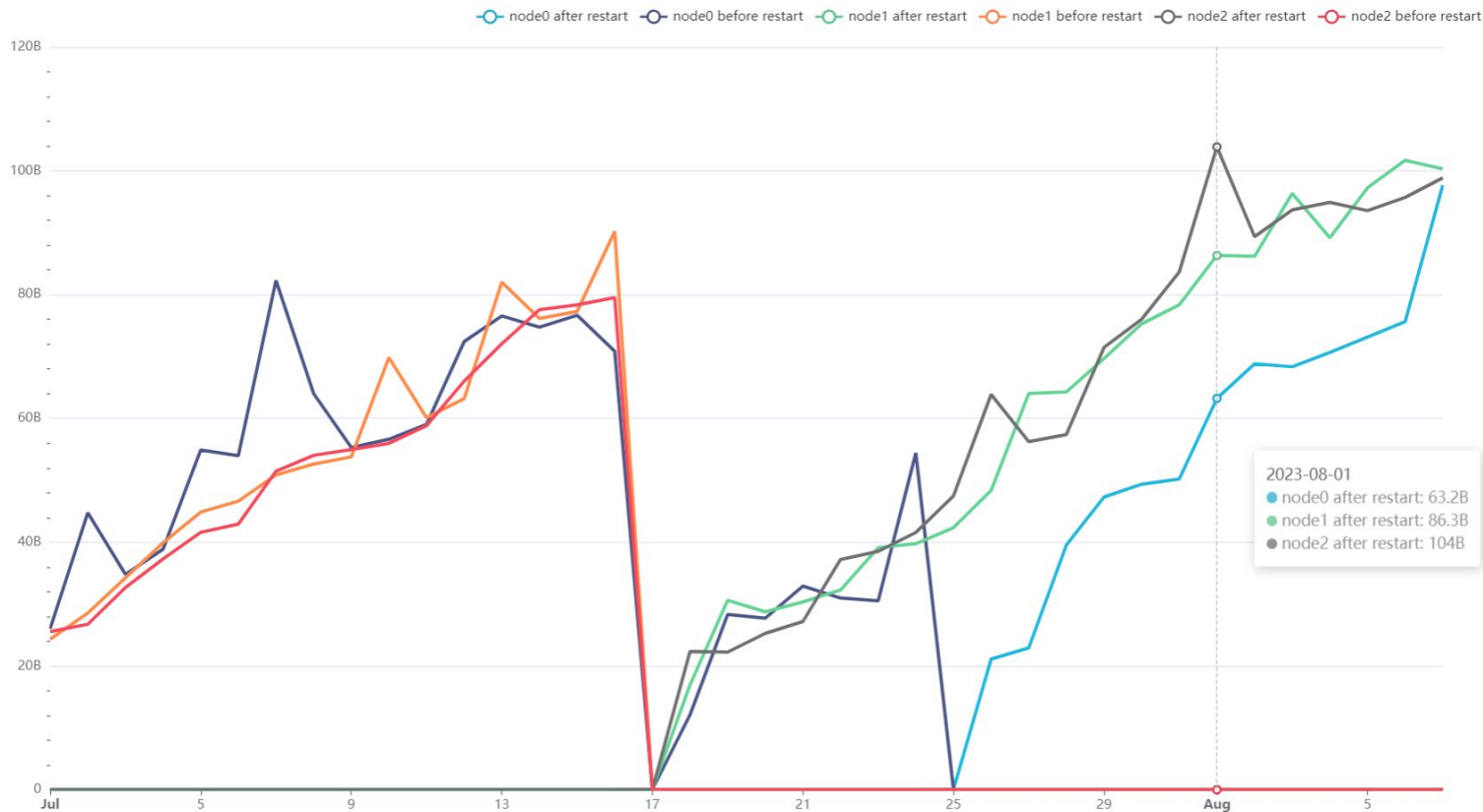
Memory leak (min MemoryTracking)

☆  **Altered**

38 rows

cached 

00:00:00.28



System tables

Database **system** contains ~100 tables with all kind of introspection

Documentation <https://clickhouse.com/docs/en/operations/system-tables>

```
SELECT
    formatReadableSize(memory_usage),
    formatReadableSize(ProfileEvents['ReadCompressedBytes']) AS ReadCompressedBytes
FROM system.query_log
WHERE (query_id = '6eb428d1-c9b2-4c4b-89fa-069383999edb') AND (type = 'QueryFinish')
```

Query id: 0923359e-2261-45b9-b46a-cf33a41e4d2a

formatReadableSize(memory_usage)	ReadCompressedBytes
51.67 GiB	124.45 MiB



Mark cache

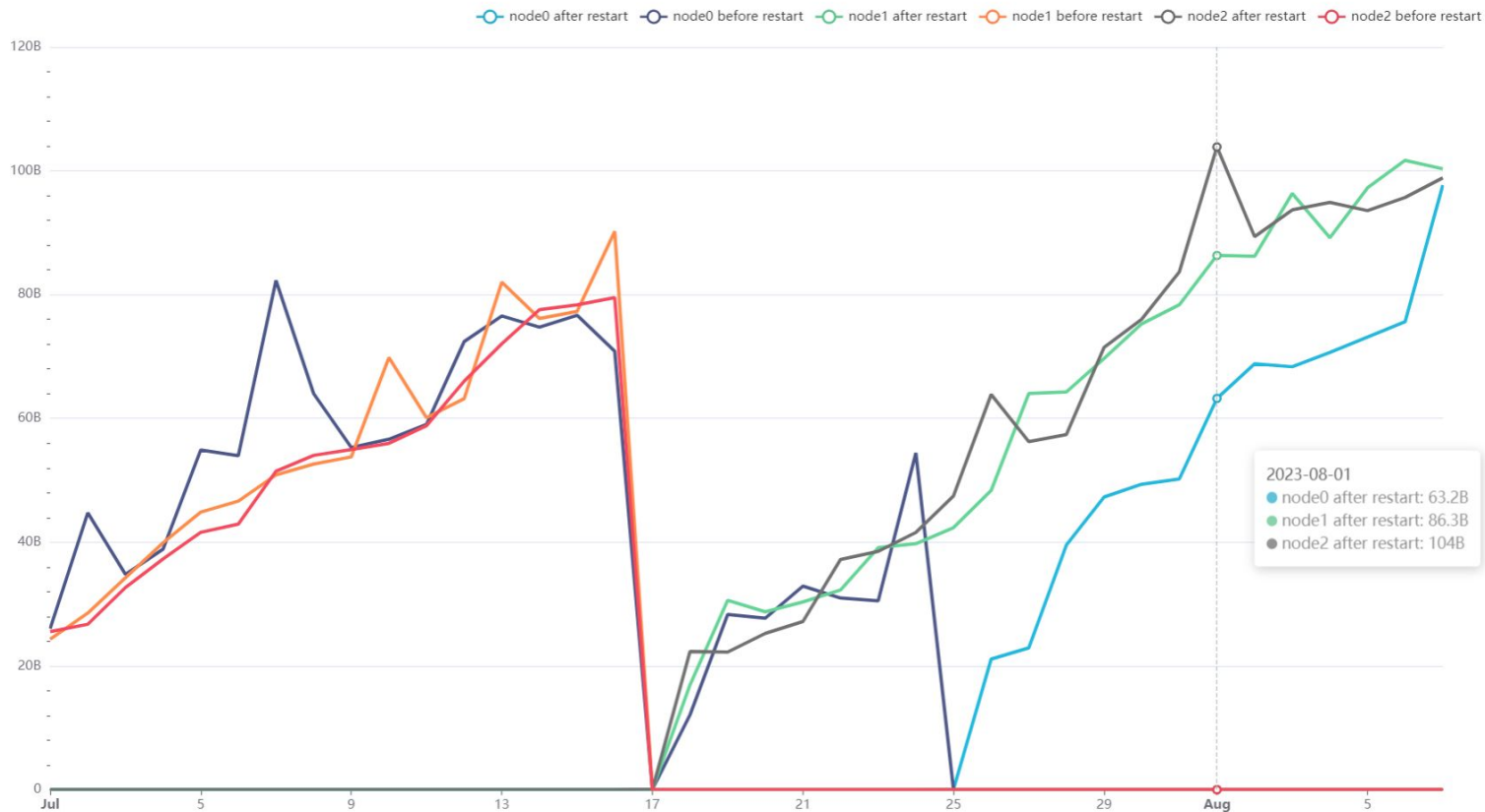
Memory leak (min MemoryTracking)

☆  **Altered**

38 rows

cached 

00:00:00.28



System tables

```
SELECT metric, formatReadableSize(value)
FROM system.metrics
WHERE metric LIKE '%MemoryTracking%'
```

metric	formatReadableSize(value)
MemoryTracking	51.01 GiB
MergesMutationsMemoryTracking	3.46 GiB

```
SELECT metric, value, formatReadableSize(value)
FROM system.asynchronous_metrics
WHERE metric LIKE 'MarkCache%'
```

metric	value	formatReadableSize(value)
MarkCacheFiles	95703	93.46 KiB
MarkCacheBytes	10737330976	10.00 GiB



Misconfiguration and Misusage

- Invalid partitioning
- `parts_to_throw_insert, max_parts_in_total, max_partitions_per_insert_block`
- `max_server_memory_usage, max_server_memory_usage_to_ram_ratio`
- High insertion rate (except `async_insert`)
- Heavy background operations
- Frequent mutations, `OPTIMIZE FINAL`



http://localhost:8123/dashboard

rounding: 60

seconds: 86400

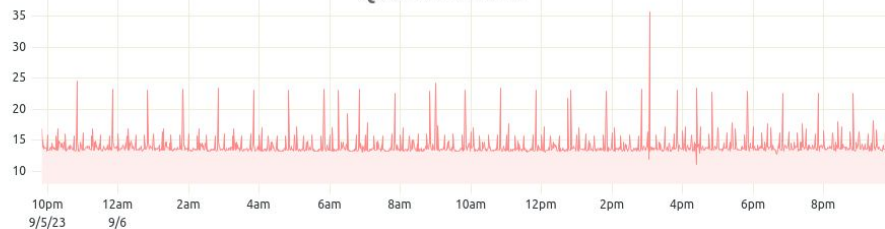
Ok

Reload

Add chart



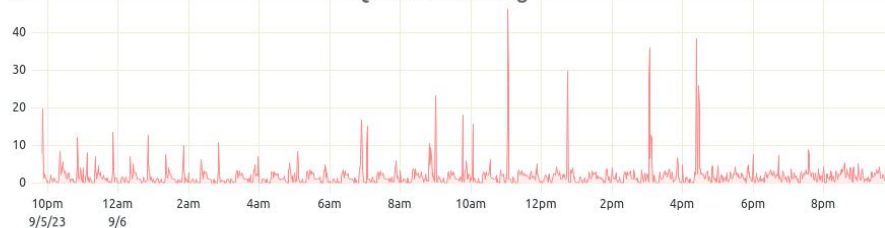
Queries/second



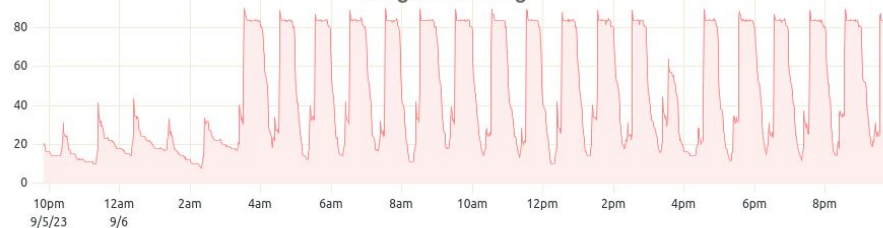
CPU Usage (cores)



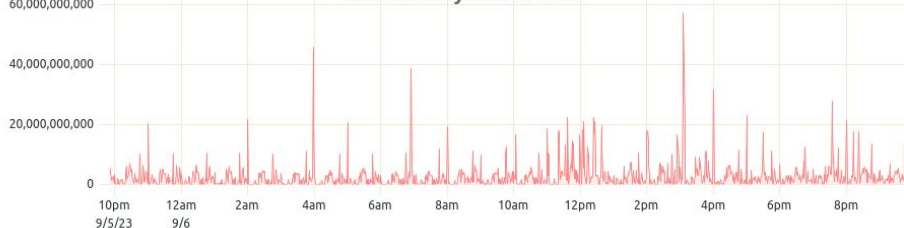
Queries Running



Merges Running



Selected Bytes/second

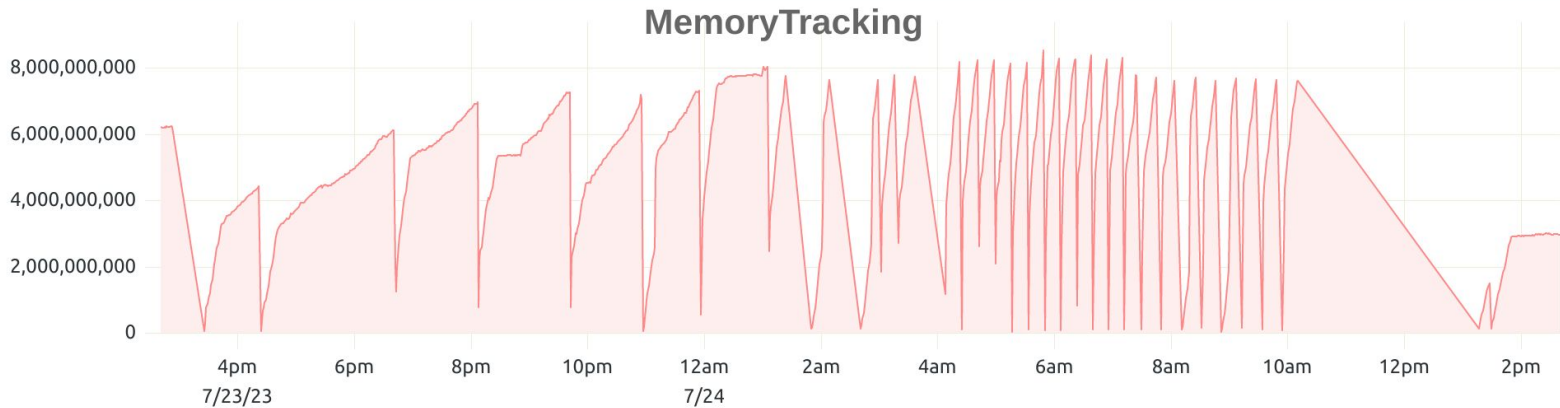
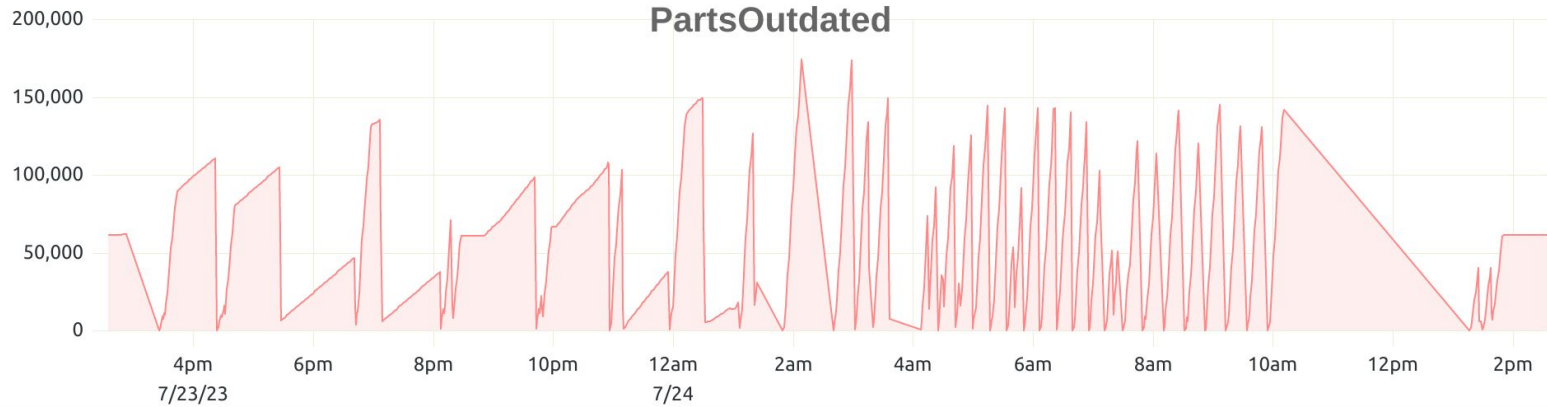


```
SELECT
  toStartOfInterval(event_time, INTERVAL {rounding:UInt32} SECOND)::INT AS t,
  avg(metric)
FROM (
  SELECT event_time, sum(ProfileEvent OSIOWaitMicroseconds) / 1000000 AS metric
  FROM clusterAllReplicas(default, system.metric_log)
  WHERE event_date >= toDate(now())
  AND event_time >= now() - {seconds:UInt32}
  GROUP BY event_time)
GROUP BY t
ORDER BY t WITH FILL STEP {rounding:UInt32} SETTINGS skip_unavailable_shards = 1
```

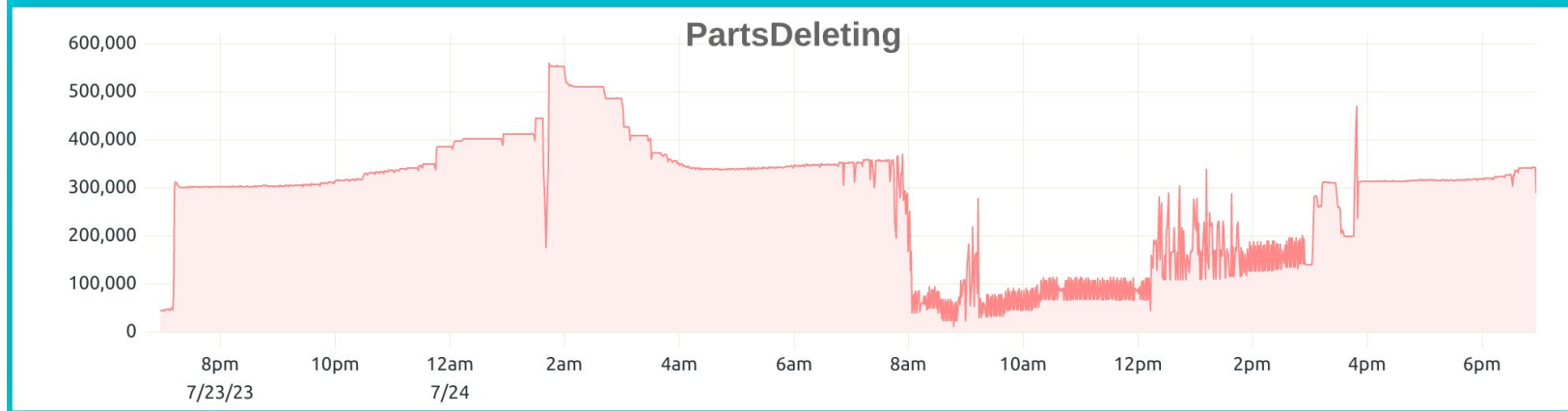
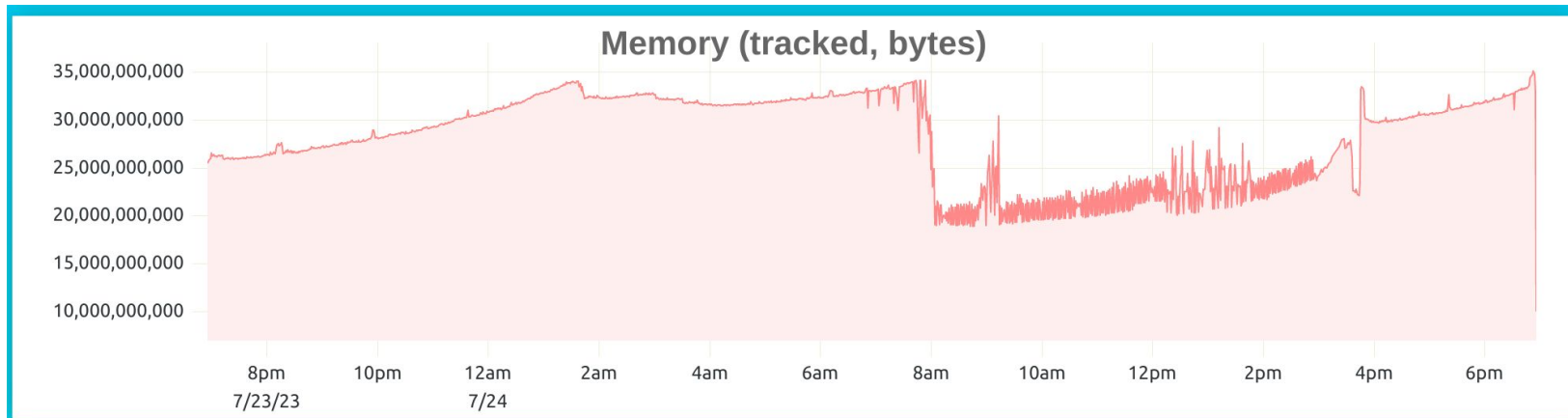
IO Wait (local fs)

Ok

Many Outdated Parts



Many Deleting Parts



Understanding of Memory Consumption

- Poor usability (currently)
- Metrics and finding metrics correlation helps a lot
- Current activity

`system.processes, system.user_processes, system.merges,
system.metrics, system.asynchronous_metrics`

- Historical activity

`system.metric_log, system.asynchronous_metric_log,
system.query_log, system.text_log`



Asynchronous Insert Memory Drift

Run a query with asynchronous insert 100 times with N = 1 million rows

```
INSERT INTO async_table SETTINGS async_insert=1, wait_for_async_insert=1 VALUES (1), ..., (N)
```

User Memory usage (before)

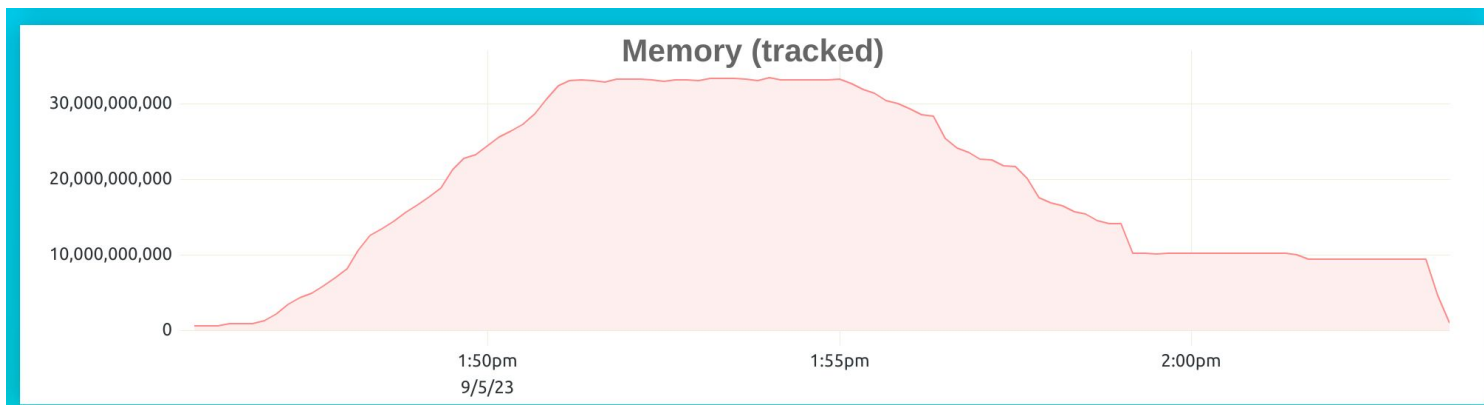
```
SELECT formatReadableSize(memory_usage)
FROM system.user_processes
WHERE user = 'A'
```

```
formatReadableSize(memory_usage)
32.70 KiB
```

User Memory usage (after)

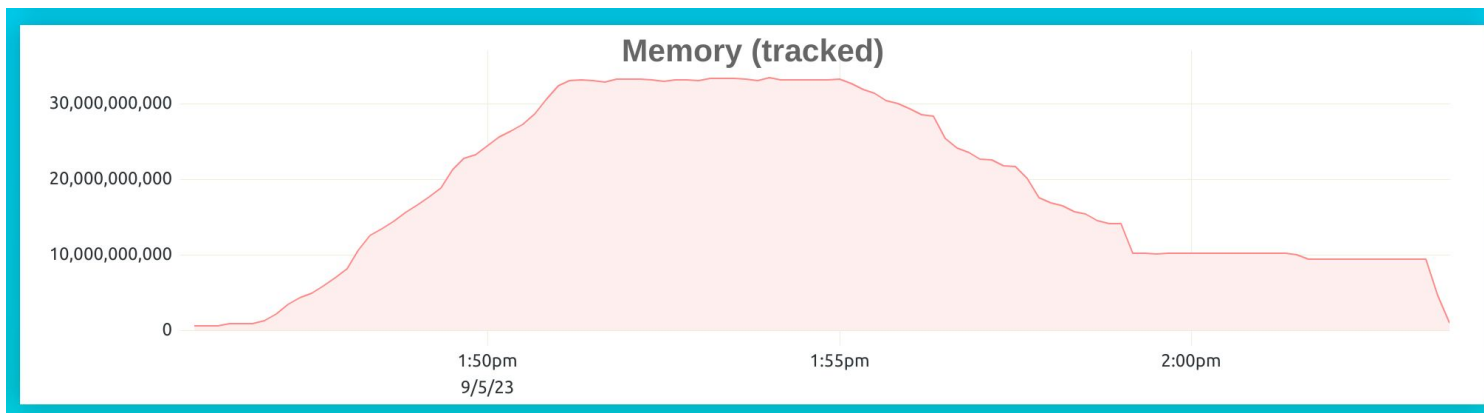
```
SELECT formatReadableSize(memory_usage)
FROM system.user_processes
WHERE user = 'A'
```

```
formatReadableSize(memory_usage)
12.21 GiB
```



Asynchronous Insert Memory Drift

- Found (mainly) by checking hypothesis and reading the code
- Fixed [#46622](#)
- Better introspection is required



Tracking Allocations

```
SELECT
    query_id,
    trace,
    size,
    ptr
FROM system.trace_log
WHERE (trace_type = 'MemorySample') AND (query_id != '') AND (size > 1000000)
LIMIT 1
```

Query id: 9a555d30-2f8e-49a9-8349-94875ebfc2a8

Row 1:

```
query_id: test_async_insert_2
trace:
[177950546,177832181,177581809,292324046,292355899,292360407,292360640,271187286,271208959,284578095,2845923
13,285024728,302263591,302264845,302681767,302672578,139962546288137,139962545389875]
size:      1572864
ptr:      139955000020096
```



Allocation StackTrace

```
SELECT arrayStringConcat(arrayMap(x -> demangle(addressToSymbol(x)), trace), '\n')  
SETTINGS allow_introspection_functions = 1
```

```
AllocationTrace::onAllocImpl(void*, unsigned long) const  
operator new(unsigned long)  
DB::Tokens::Tokens(char const*, char const*, unsigned long, bool)  
DB::tryParseQuery(...)  
DB::parseQueryAndMovePosition(...)  
DB::executeQueryImpl(...)  
DB::executeQuery(...)  
DB::HTTPHandler::processQuery(...)  
DB::HTTPHandler::handleRequest(DB::HTTPServerRequest&, DB::HTTPServerResponse&)  
DB::HTTPServerConnection::run()  
Poco::Net::TCPServerConnection::start()  
Poco::Net::TCPServerDispatcher::run()  
Poco::PooledThread::run()  
Poco::ThreadImpl::runnableEntry(void*)
```



FlameGraphs

Query example (from the benchmark)

```
SET memory_profiler_sample_probability=1, max_untracked_memory=1;

SELECT SearchPhrase, COUNTDistinct(UserID) AS u
FROM hits WHERE SearchPhrase != ''
GROUP BY SearchPhrase ORDER BY u DESC LIMIT 10
```

Use `flamegraph.pl` from <https://github.com/brendangregg/FlameGraph> and `flameGraph` function

```
SELECT arrayJoin(flameGraph(trace, size)) FROM system.trace_log WHERE trace_type = 'MemorySample' AND query_id = '...'

output | ./FlameGraph/flamegraph.pl --countname=bytes --color=mem > flame_mem.svg
```

Examples are in [this PR](#)



All Query Allocations

```
SELECT arrayJoin(flameGraph(trace, size)) FROM system.trace_log WHERE trace_type = 'MemorySample' AND query_id = '..'
```

Some memory is used to build hash table for aggregation

```
0x00007f74c135a133
0x00007f74c1435609
void* std::__1::__thread_proxy[abi:v15000]<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct, std::__1::default_delete<std::__1::__thread_struct>>, void ThreadPoolImpl<std::__1::thread>::ThreadPoolImpl<std::__1::thread>::worker(std::__1::__list_iterator<std::__1::thread, void*>)
ThreadFromGlobalPoolImpl<false>::ThreadFromGlobalPoolImpl<void ThreadPoolImpl<ThreadFromGlobalPoolImpl<false>>::scheduleImpl<void>(std::__1::function<void ()>, Priority, std::__1::option
ThreadPoolImpl<ThreadFromGlobalPoolImpl<false>>::worker(std::__1::__list_iterator<ThreadFromGlobalPoolImpl<false>, void*>)
void std::__1::__function::__policy_invoker<void ()>::__call_impl<std::__1::__function::__default_alloc_func<DB::PipelineExecutor::spawnThreads()::$_0, void ()>>(std::__1::__function::__policy_sto
DB::PipelineExecutor::executeStepImpl(unsigned long, std::__1::atomic<bool>*)
DB::ExecutionThreadContext::executeTask()
DB::AggregatingTransform::work()
DB::AggregatingTransform::consume(DB::Chunk)
DB::Aggregator::executeOnBlock(std::__1::vector<COW<DB::IColumn>::immutable_ptr<DB::IColumn>, std::__1::allocator<COW<DB::IColumn>::immutable_ptr<DB::IColumn>>>, unsigned long, u
DB::AggregatedDataVariants::convertToTwoLevel()
TwoLevelStringHashTable<StringHashMapSubMaps<char*, Allocator<true, true>>, StringHashMap<char*, Allocator<true, true>>, 8ul>::TwoLevelStringHashTable<StringHashMap<char*, Allocator<tr
StringHashTable<StringHashMapSubMaps<char*, Allocator<true, true>>>::StringHashTable()
HashTable<StringKey24, StringHashMapCell<StringKey24, char*>, StringHashTableHash, StringHashTableGrower<8ul>, Allocator<true, true>>::alloc(StringHashTableGrower<8ul> const&)
Allocator<true, true>::alloc(unsigned long, unsigned long)
AllocationTrace::onAllocImpl(void*, unsigned long) const
StackTrace::tryCapture()
all
```

Function: DB::AggregatingTransform::consume(DB::Chunk) (4,382,720 bytes, 19.55%)

FlameGraphs

Example: enable uncompressed cache

```
SET memory_profiler_sample_probability=1, max_untracked_memory=1, use_uncompressed_cache=1,  
merge_tree_max_rows_to_use_cache=100000000000, merge_tree_max_bytes_to_use_cache=100000000000;
```

```
SELECT SearchPhrase, COUNTDistinct(UserID) AS u  
FROM hits WHERE SearchPhrase != ''  
GROUP BY SearchPhrase ORDER BY u DESC LIMIT 10;
```

Now, query allocated and not released 100M

```
SELECT formatReadableSize(sum(size)) FROM system.trace_log WHERE (query_id = '..') AND (trace_type = 'MemorySample')
```

```
formatReadableSize(sum(size))  
108.90 MiB
```

Find only those allocations which were not released by query

```
SELECT arrayJoin(flameGraph(trace, size, ptr)) FROM system.trace_log WHERE trace_type = 'MemorySample' AND query_id = '...
```



Unreleased Query Allocations

Memory is hold by uncompressed cache

0x00007f74c135a133	0x00007f74c135a133	0x00007f74c135a133	0x00007f7...
0x00007f74c1435609	0x00007f74c1435609	0x00007f74c1435609	0x00007f7...
void* std::__1::__thread_proxy[abi:v15000]<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct, std::...	void* std::__1::__thread_proxy[abi:v15000]<std::__1::tuple<std::__1::u...	void* std::__1::__thread_proxy[abi...	void* std::__...
ThreadPoolImpl<std::__1::thread>::worker(std::__1::__list_iterator<std::__1::thread, void*>)	ThreadPoolImpl<std::__1::thread>::worker(std::__1::__list_iterator<std...	ThreadPoolImpl<std::__1::thread>..	ThreadPool...
ThreadFromGlobalPoolImpl<false>::ThreadFromGlobalPoolImpl<void ThreadPoolImpl<ThreadFromGlobalPool...	ThreadFromGlobalPoolImpl<false>::ThreadFromGlobalPoolImpl<void..	ThreadFromGlobalPoolImpl<false>..	ThreadFr...
ThreadPoolImpl<ThreadFromGlobalPoolImpl<false>::worker(std::__1::__list_iterator<ThreadFromGlobalPool...	ThreadPoolImpl<ThreadFromGlobalPoolImpl<false>::worker(std::__1..	ThreadPoolImpl<ThreadFromGlob...	ThreadPool...
void std::__1::__function::__policy_invoker<void ()>::__call_impl<std::__1::__function::__default_alloc_func<...	void std::__1::__function::__policy_invoker<void ()>::__call_impl<std...	void std::__1::__function::__policy...	void std::__...
DB::PipelineExecutor::executeStepImpl(unsigned long, std::__1::atomic<bool>*)	DB::PipelineExecutor::executeStepImpl(unsigned long, std::__1::atomi...	DB::PipelineExecutor::executeSte...	DB::Pipeli...
DB::ExecutionThreadContext::executeTask()	DB::ExecutionThreadContext::executeTask()	DB::ExecutionThreadContext::exec...	DB::Execut...
DB::ISource::work()	DB::ISource::work()	DB::ISource::work()	DB::ISourc...
DB::MergeTreeSource::tryGenerate()	DB::MergeTreeSource::tryGenerate()	DB::MergeTreeSource::tryGenerate()	DB::Merge...
DB::IMergeTreeSelectAlgorithm::read()	DB::IMergeTreeSelectAlgorithm::read()	DB::IMergeTreeSelectAlgorithm::r...	DB::IMerg...
DB::IMergeTreeSelectAlgorithm::readFromPart()	DB::IMergeTreeSelectAlgorithm::readFromPart()	DB::IMergeTreeSelectAlgorithm::r...	DB::IMerg...
DB::IMergeTreeSelectAlgorithm::readFromPartImpl()	DB::IMergeTreeSelectAlgorithm::readFromPartImpl()	DB::IMergeTreeSelectAlgorithm::r...	DB::IMerg...
DB::MergeTreeRangeReader::read(unsigned long, DB::MarkRanges&)	DB::MergeTreeRangeReader::read(unsigned long, DB::MarkRanges&)	DB::MergeTreeRangeReader::read...	DB::Merge...
DB::MergeTreeRangeReader::continueReadingChain(DB::MergeTreeRangeReader::ReadResult const&, unsigned l...	DB::MergeTreeRangeReader::read(unsigned long, DB::MarkRanges&)	DB::MergeTreeRangeReader::read...	DB::Merge...
DB::MergeTreeRangeReader::DelayedStream::finalize(std::__1::vector<COW<DB::IColumn>::immutable_ptr<...	DB::MergeTreeRangeReader::startReadingChain(unsigned long, DB::M...	DB::MergeTreeRangeReader::start...	DB::Merge...
DB::MergeTreeReaderWide::readRows(unsigned long, bool, unsigned long, std::__1::vector<CO...	DB::MergeTreeRangeReader::DelayedStream::finalize(std::__1::vector...	DB::MergeTreeRangeReader::Dela...	DB::Merge...
DB::MergeTreeReaderWide::readData(DB::NameAndTypePair const&, std::__1::shared_ptr<DB::ISerialization c...	DB::MergeTreeReaderWide::readRows(unsigned long, unsigned long, ..	DB::MergeTreeReaderWide::read...	DB::Merge...
DB::ISerialization::deserializeBinaryBulkWithMultipleStreams(COW<DB::IColumn>::immutable_ptr<DB::ICo...	DB::MergeTreeReaderWide::readData(DB::NameAndTypePair const&, ..	DB::MergeTreeReaderWide::readD...	DB::Merge...
DB::SerializationNumber<unsigned long>::deserializeBinaryBulk(DB::IColumn&, DB::ReadBuffer&, unsigned l...	DB::ISerialization::deserializeBinaryBulkWithMultipleStreams(COW...	DB::ISerialization::deserializeBina...	DB::ISerial...
DB::ReadBuffer::readBig(char*, unsigned long)	void DB::deserializeBinarySSE2<1>(<DB::PODArray<char8_t, 4096ul, ...	void DB::deserializeBinarySSE2<2...	void DB::d...
DB::CachedCompressedReadBuffer::nextImpl()			
DB::Memory<Allocator<false, false>::alloc(unsigned long)			
Allocator<false, false>::alloc(unsigned long, unsigned long)			
AllocationTrace::onAllocImpl(void*, unsigned long) const			
StackTrace::tryCapture()			
all			

Function: DB::CachedCompressedReadBuffer::nextImpl() (109,290,332 bytes, 89.47%)

Peak of Memory Usage

```
SELECT
    toStartOfInterval(event_time_microseconds, toIntervalMillisecond(500)) AS t,
    max(sum) AS memory,
    formatReadableSize(memory)
FROM
(
    SELECT
        event_time_microseconds,
        sum(size) OVER (ORDER BY event_time_microseconds ASC) AS sum
    FROM system.trace_log
    WHERE (query_id = '..') AND (trace_type = 'MemorySample')
)
GROUP BY t
ORDER BY t ASC
```

t	memory	formatReadableSize(memory)
2023-09-04 20:11:33.500	98272	95.97 KiB
2023-09-04 20:11:34.000	246872	241.09 KiB
2023-09-04 20:11:34.500	555200	542.19 KiB
2023-09-04 20:11:35.000	268265964	255.84 MiB
2023-09-04 20:11:35.500	284551358	271.37 MiB
2023-09-04 20:11:36.000	149204233	142.29 MiB



Query Allocations Snapshot

```
SELECT arrayJoin(flameGraph(trace, size, ptr)) FROM system.trace_log
WHERE trace_type = 'MemorySample' AND query_id = '..' AND event_time < 'timestamp'
```

0x00007f74c135a133

0x00007f74c1435609

void* std::__1::__thread_proxy[abi:v15000]<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct, std::__1::default_delete<std::__1::__thread_struct>>, void ThreadPool

ThreadPoolImpl<std::__1::thread>::worker(std::__1::__list_iterator<std::__1::thread, void*>)

ThreadFromGlobalPoolImpl<false>::ThreadFromGlobalPoolImpl<void ThreadPoolImpl<ThreadFromGlobalPoolImpl<false>>::scheduleImpl<void>(std::__1::function<void ()>, P

ThreadPoolImpl<ThreadFromGlobalPoolImpl<false>>::worker(std::__1::__list_iterator<ThreadFromGlobalPoolImpl<false>, void*>)

void std::__1::__function::__policy_invoker<void ()>::__call_impl<std::__1::__function::__default_alloc_func<DB::PipelineExecutor::spawnThreads()::\$_0, void ()>>(std::__1::

DB::PipelineExecutor::executeStepImpl(unsigned long, std::__1::atomic<bool>*)

DB::ExecutionThreadContext::executeTask()

DB::AggregatingTransform::work()

DB::AggregatingTransform::consume(DB::Chunk)

DB::Aggregator::executeOnBlock(std::__1::vector<COW<DB::IColumn>::immutable_ptr<DB::IColumn>, std::__1::allocator<COW<DB::IColumn>::immutable_ptr<DB::IColum

DB::Aggregator::executeImpl(DB::AggregatedDataVariants&, unsigned long, unsigned long, std::__1::vector<DB::IColumn const*, std::__1::allocator<DB::IColumn const*>>&, I

void DB::Aggregator::executeImplBatch<false, false, false, DB::AggregationMethodStringNoCache<TwoLevelStringHashMap<char*, Allocator<true, true>, StringHashMap>, fals

DB::Arena::addMemoryChunk(unsigned long)

Allocator<false, false>::alloc(unsigned long, unsigned long)

AllocationTrace::onAllocImpl(void*, unsigned long) const

StackTrace::tryCapture()

all

Function: DB::Arena::addMemoryChunk(unsigned long) (112,050,176 bytes, 42.17%)

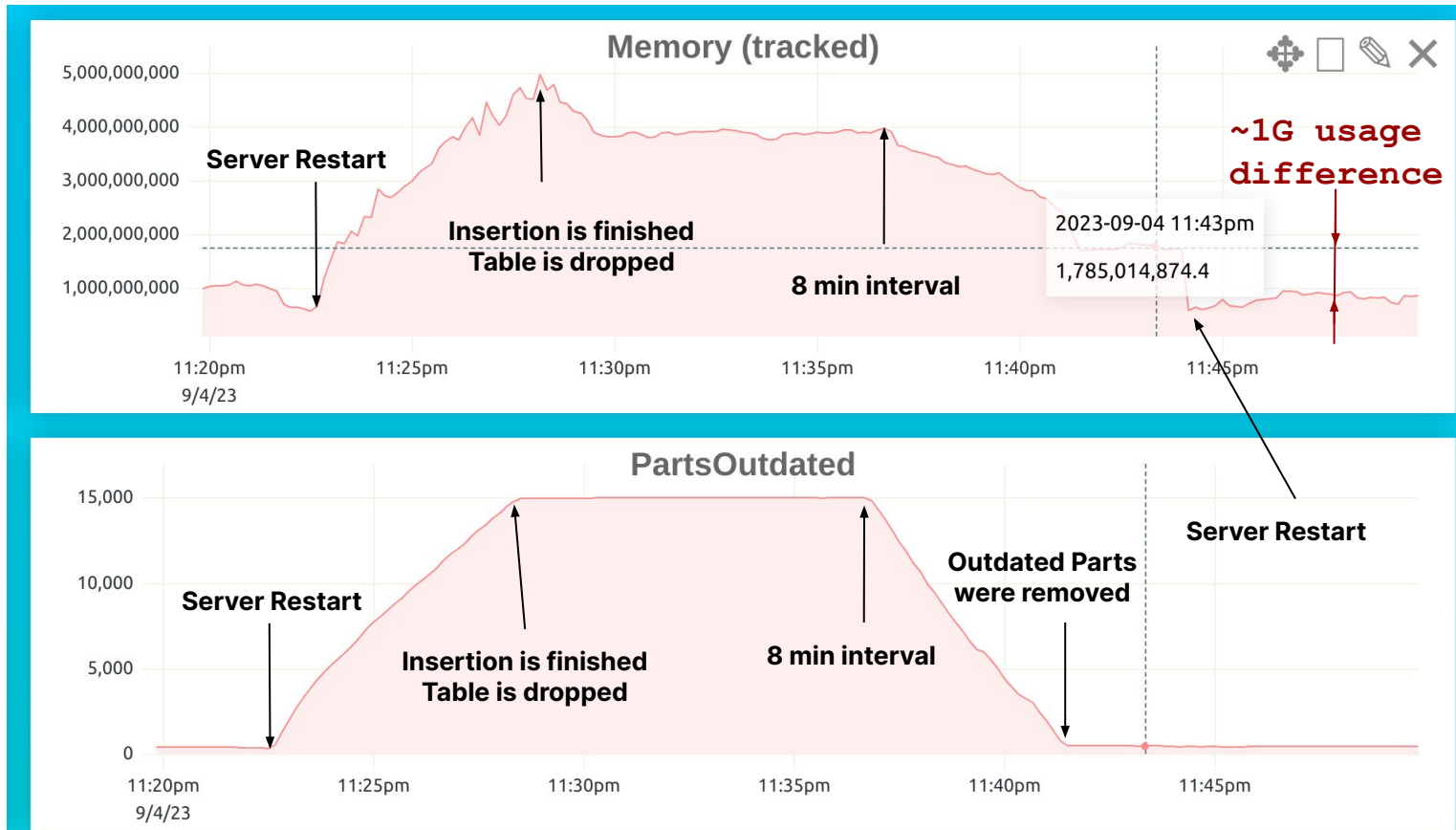
Query Deallocations Snapshot

```
SELECT arrayJoin(flameGraph(trace, size, -ptr)) FROM system.trace_log
WHERE trace_type = 'MemorySample' AND query_id = '..' AND event_time >= 'timestamp'
```

```
0x000.. 0x00007f74c135a133
0x000.. 0x00007f74c1435609
void* .. void* std::__1::__thread_proxy[abi:v15000]<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct, std::__1::default_delete<std::__1::__thread_struct>>, void
Threa.. ThreadPoolImpl<std::__1::thread>::worker(std::__1::__list_iterator<std::__1::thread, void*>)
Threa.. ThreadFromGlobalPoolImpl<false>::ThreadFromGlobalPoolImpl<void ThreadFromGlobalPoolImpl<ThreadFromGlobalPoolImpl<false>>::scheduleImpl<void>(std::__1::function
Threa.. ThreadPoolImpl<ThreadFromGlobalPoolImpl<false>>::worker(std::__1::__list_iterator<ThreadFromGlobalPoolImpl<false>, void*>)
void std.. void std::__1::__function::__policy_invoker<void ()>::__call_impl<std::__1::__function::__default_alloc_func<DB::PipelineExecutor::spawnThreads()::$_0, void ()>>
DB::P.. DB::PipelineExecutor::executeStepImpl(unsigned long, std::__1::atomic<bool>*)
DB::E.. DB::ExecutionContext::executeTask()
DB::L.. DB::ISource::work()
DB::L.. DB::ISource::tryGenerate()
DB:... DB::ConvertingAggregatedToChunksWithMergingSource::generate()
DB:... DB::Aggregator::mergeAndConvertOneBucketToBlock(std::__1::vector<std::__1::shared_ptr<DB::AggregatedDataVariants>, std::__1::allocator<std::__1::shared_ptr<
std::__1::vector<std::__1::shared_ptr<DB::AggregatedDataVariants>, std::__1::allocator<std::__1::shared_ptr<DB::AggregatedDataVariants>>>, false>, false>
StringHashTable<StringHashMapSubMaps<char*, Allocator<true, true>>>::clearAndShrink()
Allocator<true, true>::free(void*, unsigned long)
AllocationTrace::onFreeImpl(void*, unsigned long) const
StackTrace::tryCapture()
all
```

Function: DB::ConvertingAggregatedToChunksWithMergingSource::generate() (115,834,880 bytes, 43.60%)

Insert + Drop test



Insert + Drop test

Hard to debug with memory profiler

- A lot of allocations
- `system.trace_log` traces itself
- High memory consumption was detected in narrow size class

More introspection features

- Settings `memory_profiler_sample_min/max_allocation_size` to limit tracked allocations size
- Table `system.jemalloc_bins` to show statistics from jemalloc arenas

Fix [#51732](#)



Summary

- Metrics correlation is useful
- System tables can show current and historical activity
- Recent new features
 - ◇ [system.user_processes](#)
 - ◇ [memory_profiler_sample_min/max_allocation_size](#)
 - ◇ [system.jemalloc_bins](#)
 - ◆ [flameGraph](#)
- With `system.trace_log` we can build a snapshot of query allocations at any point of time
- Flamegraphs are built with an external script
- Need to support flamegraphs it in `/dashboard`



Thank You!

