ClickHouse performance optimizations

About me

Maksim, developer of ClickHouse.

Performance of ClickHouse

1. High Level System Architecture.

2. CI/CD Pipeline.

3. Introspection.

4. Abstractions and Algorithms.

5. Libraries.

6. JIT compilation. Dynamic dispatch.

High Level System Architecture

ClickHouse Architecture

Column-oriented storage — data is physically stored by columns.

Only necessary columns are read from disk during query.

Better compression because of data locality.

Vectorized Query Execution

Vectorized query execution — data is processed in blocks. Block contains multiple columns with **max_block_size** rows (65505 by default).

Each column is stored as a vector of primitive data types or their combination:

1. Better utilization for CPU caches and pipeline.

2. Data is processed using SIMD instructions.

ClickHouse Columns

Numeric columns — **PODArray**. Almost the same as **std::vector**.

Nullable columns contain data column and UInt8 column bitmask is element null.

Array columns contain data column and UInt64 column with offsets.

Const column contain 1 constant value.

CI/CD Pipeline

CI/CD Pipeline

1. Functional, Integration tests.
2. Run all tests with sanitizers (ASAN, TSAN, MSAN).
3. Fuzzers (data types, compression codecs).
4. AST fuzzer.
5. Stress tests (Our special TSAN stress).
6. **Performance tests**.

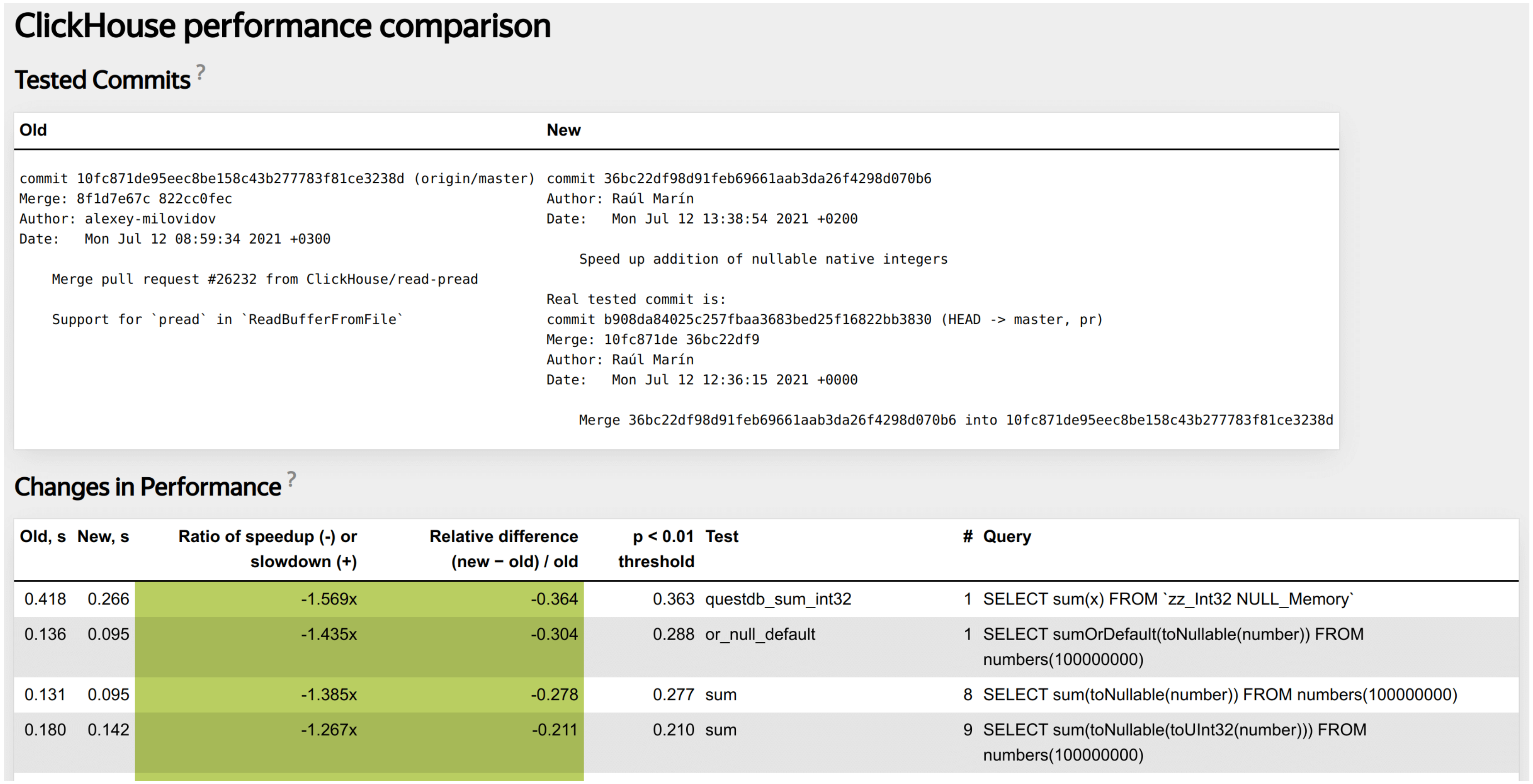
Performance Tests

Part of CI/CD pipeline.

Runs for each commit in pull request.

Runs for each commit in the master branch.

Performance Tests



ClickHouse performance comparison

Tested Commits ?

Old	New
commit 10fc871de95eec8be158c43b277783f81ce3238d (origin/master) Merge: 8f1d7e67c 822cc0fec Author: alexey-milovidov Date: Mon Jul 12 08:59:34 2021 +0300 Merge pull request #26232 from ClickHouse/read-pread Support for \`pread\` in \`ReadBufferFromFile\`	commit 36bc22df98d91feb69661aab3da26f4298d070b6 Author: Raúl Marín Date: Mon Jul 12 13:38:54 2021 +0200 Speed up addition of nullable native integers Real tested commit is: commit b908da84025c257fbaa3683bed25f16822bb3830 (HEAD -> master, pr) Merge: 10fc871de 36bc22df9 Author: Raúl Marín Date: Mon Jul 12 12:36:15 2021 +0000 Merge 36bc22df98d91feb69661aab3da26f4298d070b6 into 10fc871de95eec8be158c43b277783f81ce3238d

Changes in Performance ?

Old, s	New, s	Ratio of speedup (-) or slowdown (+)	Relative difference (new − old) / old	p < 0.01 threshold	Test	#	Query
0.418	0.266	-1.569x	-0.364	0.363	questdb_sum_int32	1	SELECT sum(x) FROM \`zz_Int32 NULL_Memory\`
0.136	0.095	-1.435x	-0.304	0.288	or_null_default	1	SELECT sumOrDefault(toNullable(number)) FROM numbers(100000000)
0.131	0.095	-1.385x	-0.278	0.277	sum	8	SELECT sum(toNullable(number)) FROM numbers(100000000)
0.180	0.142	-1.267x	-0.211	0.210	sum	9	SELECT sum(toNullable(toUInt32(number))) FROM numbers(100000000)

Performance Tests

Collect different statistics during each performance test run. Can be useful for later debugging.

Processor metrics (CPU cycles, cache misses same as perf-stat).

ClickHouse specific profile events (read bytes from disk, transferred over network, etc).

https://clickhouse.com/blog/testing-the-performance-of-click-house/

Performance Tests

Helps us find performance regressions.

Nice tool that can help to find places where performance can be improved.

1. Try different allocators, different libraries.

2. Try different compiler options (loop unrolling, inline threshold)

3. Enable **AVX/AVX2/AVX512** for build.

Introspection

Basic Introspection

Collect **ProfileEvents** for each query:

RealTimeMicroseconds, UserTimeMicroseconds, SystemTimeMicroseconds, SoftPageFaults, HardPageFaults using **getrusage** system call.

Collect **:taskstats** from procFS.

OSCPUVirtualTimeMicroseconds, OSCPUWaitMicroseconds (when **/proc/thread-self/schedstat** is available). OSIOWaitMicroseconds (when **/proc/thread-self/stat** is available). OSReadChars, OSWriteChars, OSReadBytes, OSWriteBytes (when **/proc/thread-self/io** is available)

https://man7.org/linux/man-pages/man2/getrusage.2.html

https://man7.org/linux/man-pages/man5/proc.5.html

Basic Introspection

Collect **ProfileEvents** for each query:

Hardware specific counters CPU cache misses, CPU branch mispredictions using **perf_event_open** system call.

https://man7.org/linux/man-pages/man2/perf_event_open.2.html

Basic Introspection

Collect **ProfileEvents** for each query:

Different ClickHouse specific metrics FileOpen, DiskReadElapsedMicroseconds, NetworkSendBytes.

Example Basic Introspection

```
SELECT PE.Names AS ProfileEventName, PE.Values AS ProfileEventValue
FROM system.query_log ARRAY JOIN ProfileEvents AS PE
WHERE query_id='344b07d9-9d7a-48f0-a17e-6f5f6f3d61f5'
AND ProfileEventName LIKE 'Perf%';
```

```
┌─ProfileEventName───────────────┬─ProfileEventValue─┐
│ PerfCpuCycles                  │       40496995274 │
│ PerfInstructions               │       57259199973 │
│ PerfCacheReferences            │        2072274618 │
│ PerfCacheMisses                │         146570206 │
│ PerfBranchInstructions         │        8675194991 │
│ PerfBranchMisses               │         259531879 │
│ PerfStalledCyclesFrontend      │         813419527 │
│ PerfStalledCyclesBackend       │       15797162832 │
│ PerfCpuClock                   │       10587371854 │
│ PerfTaskClock                  │       10587382785 │
│ PerfContextSwitches            │              3009 │
│ PerfCpuMigrations              │               113 │
│ PerfMinEnabledTime             │       10584952104 │
│ PerfMinEnabledRunningTime      │        4348089512 │
│ PerfDataTLBReferences          │         465992961 │
│ PerfDataTLBMisses              │           5149603 │
│ PerfInstructionTLBReferences   │           1344998 │
│ PerfInstructionTLBMisses       │            181635 │
└────────────────────────────────┴───────────────────┘
```

Stacktraces Collection

Periodically collect stack traces from all currently running threads.

Currently using a patched fork of **LLVM libunwind**.

Example Stacktraces Collection

Check all threads current stack trace from **system.stack_trace**

```
WITH arrayMap(x -> demangle(addressToSymbol(x)), trace) AS all
SELECT thread_name, thread_id, query_id, arrayStringConcat(all, '\n') AS res
FROM system.stack_trace LIMIT 1 FORMAT Vertical;
```

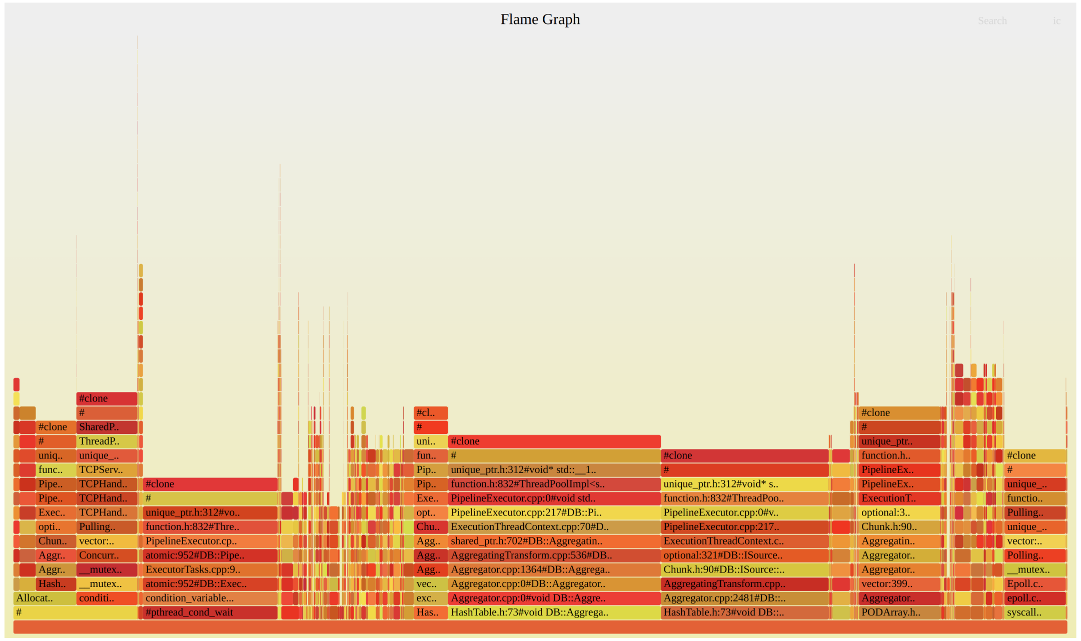
```
Row 1:
──────
thread_name: clickhouse-serv
thread_id:   125441
query_id:
res:         pthread_cond_wait
std::__1::condition_variable::wait(std::__1::unique_lock&)
BaseDaemon::waitForTerminationRequest()
DB::Server::main(/*arguments*/)
Poco::Util::Application::run()
DB::Server::run()
Poco::Util::ServerApplication::run(int, char**)
mainEntryClickHouseServer(int, char**)
main
__libc_start_main
_start
```

Example Stacktraces Flame Graph

Generate flamegraph of query execution

```
./clickhouse-client --query="SELECT
arrayStringConcat(
    arrayMap(x -> concat(
        splitByChar('/', addressToLine(x))[-1],
        '#',
        demangle(addressToSymbol(x))),
        trace),
    ';') AS stack,
count(*) AS samples
FROM system.trace_log
WHERE (trace_type = 'Real') AND (query_id = '344b07d9-9d7a-48f0-a17e-6f5f6f3d61f5')
GROUP BY trace" | flamegraph.pl
```

Example Stacktraces Flame Graph



https://www.brendangregg.com/flamegraphs.html

Abstractions and Algorithms

Abstractions and Algorithms

For high performance systems interfaces must be determined by algorithms.

Top-down approach does not work.

High-performance system must be designed concentrating on doing a single task efficiently.

Designed from hardware capabilities.

ClickHouse was designed to efficiently FILTER and GROUP BY data that fits in RAM.

https://presentations.clickhouse.com/bdtc_2019

Abstractions and Algorithms

There is no silver bullet, or best algorithm for any task.

Try to choose the fastest possible algorithm/algorithms for **your specific task**.

Performance must be evaluated on real data.

Most of the algorithms are affected by data distribution.

Abstractions and Algorithms

Complex task can be viewed as number of small tasks.

Such small tasks can also be viewed as special cases that can be optimized.

For any task there are dozens of different algorithms that can be combined together (Example Sorting, Aggregation).

Each algorithm can be tuned later using different low-level optimizations (Data layout, Specializations, SIMD instructions, JIT compilation).

Example Aggregation

High level design desigion — data must be processed not only by multiple threads, but by multiple servers.

Core component is the HashTable framework.

Different HashTable for different types of keys (Special StringHashTable for Strings).

Example Aggregation

Additional specializations for **Nullable**, **LowCardinality**

Tuned a lot of low-level details, like allocations, structures layout in memory, batch multiple operations to avoid virtual function calls.

Added JIT compilation for special cases.

Added cache of hash-table sizes.

Abstractions and Algorithms

Optimizing performance is about trying different approaches.

Most of the time without any results.

Libraries

Libraries

If someone on the Internet says my algorithm is fastest we will try it in ClickHouse.

Always try to find interesting algorithms, and solutions.

Libraries

```
ClickHouse/contrib$ ls | grep -v "cmake" | wc -l
95
```

1. Different algorithms for parsing floats, json (multiple libraries).

2. A lot of integrations.

3. Embedded storages.

4. LLVM for JIT compilation.

5. libcxx (C++ standard library).

Libraries

Almost in any library our CI system finds bugs. We report them to maintainers.

We also have a lot of library forks with a lot of changes. For example **POCO**, **LLVM libunwind**.

Libraries

We are not afraid of adding additional contrib. Our CI system will do the job.

JIT compilation. Dynamic dispatch.

JIT Compilation

JIT compilation can transform dynamic configuration into static configuration.

Not all functions can be easily compiled, not all algorithms can be easily compiled.

Has its own costs (compilation time, memory, maintenance).

But can greatly improve performance in special cases.

JIT Compilation

Compile evaluation of multiple expressions. Example: SELECT a + b * c + 5 FROM test_table;

Compile special cases for **GROUP BY**. Example: SELECT sum(a), avg(b), count(c) FROM test_table;

Compile comparator in **ORDER BY**. Example: SELECT * FROM test_table ORDER BY a, b, c;

In all cases we transform dynamic configuration into static.

My presentation from CPP Russia 2021 JIT in ClickHouse:

https://www.youtube.com/watch?v=H_pUmU-uobI

Dynamic Dispatch

ClickHouse distributed as portable binary.

We use the old instruction set **SSE4.2**.

For **AVX**, **AVX2**, **AVX512** instructions need to use runtime instructions specialization using **CPUID**.

In addition a lot of companies bring us SIMD optimizations (ContentSquare, Intel), before most such optimizations were disabled during compilation time.

It is important that compilers can vectorize even complex loops. We can rely on this.

Dynamic Dispatch

Main idea apply compiler flags to some functions, to compile it with **AVX**, **AVX2**, **AVX512**

Then in runtime check **CPUID** and execute specialized function.

Dynamic Dispatch Example

1. Improved performance of unary functions in 1.15 - 7 times.

2. Improved performance of **sum**, **avg** aggregate functions when there are no expressions in GROUP BY in 1.2 - 1.8 times.

Dynamic Dispatch Example

For AVX2 we use such optimizations a lot.

For AVX512 currently we do not apply a lot of such optimizations. It potentially could decrease performance of other system parts.

Latest Intel processors like **Rocket Lake** and **Ice Lake** fix this issue. We can detect such processors in runtime and then use optimizations.

https://stackoverflow.com/questions/56852812/simd-instructions-lowering-cpu-frequency

Conclusion

1. CI/CD infrastructure, especially performance tests, must be the core component of a high performance system.

2. Without deep introspection it is hard to investigate issues with performance.

3. For high performance systems interfaces must be determined by algorithms.

4. Add specializations for special cases.

5. Tune your performance using low-level techniques (Data layout, JIT compilation, Dynamic Dispatch).

Questions?