

The Secrets of ClickHouse Performance Optimizations

or: How To Write Efficient Code

Trivial facts

and ground truths.

Design Principles

"top-down"

or "bottom-up" design?

Design Principles

Top-Down:

- choose a high-level architecture;
- choose what classes will be in the codebase;
- draw some diagrams;



This is not optimal

Design Principles



Bottom-Up:

- how the inner loop in my code will work?
- what is the data layout in memory?
- what bytes are read/written and where?

This is more optimal

How ClickHouse was Designed

ClickHouse was developed from a prototype,
implemented in year 2008
that was intended to solve **just a single task**:
— to filter and aggregate data **as fast as possible**.

— in other words, just to do **GROUP BY**.

<http://highscalability.com/blog/2017/9/18/evolution-of-data-structures-in-yandexmetrics.html>

How ClickHouse was Designed

Построить отчёт по: ?

UTM source	▼	▲	-	+
UTM medium	▼	▲	-	+
UTM campaign	▼	▲	-	+
UTM content	▼	▲	-	+
поисковому запросу	▼	▲	-	+

Посчитать: ?

количество визитов		▼	▲	-	+
количество просмотров	сумма	▼	▲	-	+
отказ	среднее	▼	▲	-	+
продолжительность визита	среднее	▼	▲	-	+

При условиях: ?

рекламная система	равно	Директ	▼	▲	-	+
-------------------	-------	--------	---	---	---	---

Design From Hardware Capabilities

1. What are the basic numbers (throughput, latency, volume...) of our hardware (CPU, RAM, SSD, HDD, network...) on what operations?
2. What are the data structures we use? and how they work with our hardware?
3. and do some basic math...

Design From Hardware Capabilities

Example:

- we will do GROUP BY in memory;
- will put all data in a hash table;
- if the hash table is large, it will not fit in L3 cache of CPU;
- if the values of GROUP BY keys are not distributed locally, then we have L3 cache miss for every row in a table;
- L3 cache miss has 70..100 ns **latency**;

How many keys per second we can process?

Design From Hardware Capabilities

Example:

```
SELECT rand() % 100000000 AS k  
FROM system.numbers_mt  
GROUP BY k
```

175.52 million rows/s.

Design From Hardware Capabilities

L3 cache miss has throughput of 40 million ops/sec. on a single CPU core
and ~ 500 million. ops/sec*. on 32 hyper-threading CPU cores
(Xeon E5-2650v2).

Never mess up **latency** and **throughput**!

* but we have just 175 million rows per second. Is ClickHouse slow?

Algorithms First, Abstractions Go After

If you need maximum performance

— **then interfaces in the code are determined by algorithms!**

Algorithms First, Abstractions Go After

Example: substring search:

- in C: **strstr**, **memmem**;
- in C++: **std::search**, **std::string::find**.

But these functions are **slow**! (in some usage scenario).

Algorithms First, Abstractions Go After

Substring Search:

```
void * memmem(const void * haystack, size_t haystacklen,  
              const void * needle, size_t needlelen);
```

- there is no separate initialization routine;
- required to be reentrant — cannot allocate memory.

But what if we search a single **needle** in 1 000 000 different **haystacks**?

```
Searcher searcher(needle);  
for (const auto & haystack : haystacks)  
    searcher.search(haystack);
```

Algorithms First, Abstractions Go After

Substring Search:

```
void * memmem(const void * haystack, size_t haystacklen,  
              const void * needle, size_t needlelen);
```

If we search a single **needle** in 1 000 000 different **haystacks**, then neither of **strstr**, **memmem**, **std::search**, **std::string::find** will work fast, because **their interface is not suitable**.

And without changing the interface you cannot make them fast.

You Can Always Do Better!

If you know your task better.

— substring search;

(but some smart guys have already implemented **std::search**)

— array sorting;

(but some smart guys have already implemented **std::sort**)

— hash table;

(but some smart guys have already implemented **std::unordered_map**)

Every Problem is a Landscape

Substring Search:

- exact or approximate search?
- one or multiple substrings?
- the set of substrings is explicit or specified by a language?
- substrings are rather short or long?
- substring is a sequence of
bytes / unicode code points / characters with custom collation / words?
- a search in predefined text or the text is not known in advance?
- is located in memory completely or available as a stream of data?
- with strong guarantees on time or not?

Brute Force algorithm Deterministic Finite Automaton algorithm Karp-Rabin algorithm
Shift Or algorithm Morris-Pratt algorithm Knuth-Morris-Pratt algorithm
Simon algorithm Colussi algorithm Galil-Giancarlo algorithm
Apostolico-Crochemore algorithm Not So Naive algorithm Boyer-Moore algorithm
Turbo BM algorithm Apostolico-Giancarlo algorithm Reverse Colussi algorithm
Horspool algorithm Quick Search algorithm Tuned Boyer-Moore algorithm
Zhu-Takaoka algorithm Berry-Ravindran algorithm Smith algorithm Raita algorithm
Reverse Factor algorithm Turbo Reverse Factor algorithm Forward Dawg Matching algorithm
Backward Nondeterministic Dawg Matching algorithm Backward Oracle Matching algorithm
Galil-Seiferas algorithm Two Way algorithm String Matching on Ordered Alphabets algorithm
Optimal Mismatch algorithm Maximal Shift algorithm Skip Search algorithm
KMP Skip Search algorithm Alpha Skip Search algorithm

<https://www-igm.univ-mlv.fr/~lecroq/string/>

But none of these algorithms are used in ClickHouse!

Every Problem is a Landscape

What ClickHouse is using:

- **Volnitsky** algorithm when needle is constant;
- SIMD optimized brute-force for non-constant needle;
- variation of **Volnitsky** algorithm for a set of constant needles;
- re2 and hyperscan for regular expressions.

<https://habr.com/en/company/yandex/blog/466183/> (russian)

Every Problem is a Landscape

Sorting

- array of numbers / tuples / strings / structures?
- available completely in RAM?
- with comparisons / 3-way comparisons / parallel comparisons / by radix?
- direct / indirect (not sort, obtain a permutation)?
- stable / non-stable?
- full / partial / n-th element?
- finish sorting / merging / incomplete sorting?

Every Problem is a Landscape

ClickHouse is using **pdqsort** and **radix sort**,
... but it's not perfect, must rewrite.

Every Problem is a Landscape

Hash Table (my favorite)

- the choice of hash function;
- memory layout: open-addressing vs. chaining;
- small or big values;
- support for non-moveable values;
- memory layout: one array for keys and values or separate arrays;
- collision resolution algorithm;
- algorithm for values removal;
- fill factor; when and how to resize;
- how to move values around on resize;
- fast probing with bitmaps;
- inline placement of string keys;
- prefetch and batching;

Hash Table

We use the best* hash table in ClickHouse.

- * the best for our needs.

- * not a single but multiple different hash tables.

- * and we constantly trying to do better:

<https://github.com/ClickHouse/ClickHouse/pull/5417>

— Add StringHashMap to optimize string aggregation by **Amos Bird**.

You Can Always Do Better!

If you know your task better.

- substring search;

- array sorting;

- hash table;

...

- allocating memory (**malloc**);

- copying bytes around (**memcpy**);

You Can Always Do Better!

In ClickHouse we use:

— chinese memcpy:

<https://github.com/ClickHouse/ClickHouse/tree/master/libs/libmemcpy>

— a special memcpy to gather short pieces of memory:

memcpySmallAllowReadWriteOverflow15

You Can Always Do Better!

or grab the best!

If someone on the internet said that "my algorithm is the best"
— use that algorithm in ClickHouse.

... and probably throw away if it isn't the best.

Example: **simdjson** — adopted, still using.

Example: **mimalloc** — tried, throwed away.

Specialization For the Tasks

Trivial example:

```
WHERE str LIKE '%hello%world!%'
```

— regular expression (re2)

but substring search for "world!" before;

```
WHERE str LIKE '%hello%'
```

— substring search;

```
WHERE str LIKE 'hello%'
```

— prefix comparison.

But even MySQL has similar optimization.

Specialization For the Tasks

Specialization For the Data Types (example: GROUP BY):

```
using AggregatedDataWithoutKey = AggregateDataPtr;

...UInt8Key = FixedHashMap<UInt8, AggregateDataPtr>;
...UInt16Key = FixedHashMap<UInt16, AggregateDataPtr>;

...UInt64Key = HashMap<UInt64, AggregateDataPtr, HashCRC32<UInt64>>;
...StringKey = HashMapWithSavedHash<StringRef, AggregateDataPtr>;
...Keys128 = HashMap<UInt128, AggregateDataPtr, UInt128HashCRC32>;
...Keys256 = HashMap<UInt256, AggregateDataPtr, UInt256HashCRC32>;

...UInt64KeyTwoLevel = TwoLevelHashMap<UInt64, AggregateDataPtr, HashCRC32<UInt64>>;
...StringKeyTwoLevel = TwoLevelHashMapWithSavedHash<StringRef, AggregateDataPtr>;
...Keys128TwoLevel = TwoLevelHashMap<UInt128, AggregateDataPtr, UInt128HashCRC32>;
...Keys256TwoLevel = TwoLevelHashMap<UInt256, AggregateDataPtr, UInt256HashCRC32>;

...UInt64KeyHash64 = HashMap<UInt64, AggregateDataPtr, DefaultHash<UInt64>>;
...StringKeyHash64 = HashMapWithSavedHash<StringRef, AggregateDataPtr, StringRefHash64>;
...Keys128Hash64 = HashMap<UInt128, AggregateDataPtr, UInt128Hash>;
...Keys256Hash64 = HashMap<UInt256, AggregateDataPtr, UInt256Hash>;
```

— [dbms/src/Interpreters/Aggregator.h](#)

Specialization For the Tasks

Specialization for different amounts of data.

Example: **quantileTiming** function:

- less than 64 values — flat array in memory arena;
- less than 5670 values — flat array in heap memory;
- more — a histogram with custom buckets.

([QuantileTiming.h](#))

Example: **uniqCombined** function:

- flat array;
- hash table;
- HyperLogLog.

([CombinedCardinalityEstimator.h](#))

Data Structures are Always in Context of the Task

How to choose the data structure?

- find out what it implements, needed and unneeded.

Trivial example: **std::string**:

- implements memory management by itself.
- allow to modify a string,
e.g. append one more character.
- tracks string size by its own.

Data Structures are Always in Context of the Task

Trivial example: how to implement GROUP BY?

Option 1:

- sort the array by keys;
- then iterate through it,
and calculate aggregate functions for consecutive identical keys.

Option 2:

- put all keys into a hash table;
- when the key is found again,
update the states of aggregate functions.

Answer: option 2 is better; but if the data is almost sorted then better to finish sorting and apply option 1; but if the data doesn't fit in memory, partition it by buckets and then option 2.

Algorithms Know About Data Distribution

```
#ifdef __SSE2__
/** A slightly more optimized version.
 * Based on the assumption that often sequences of consecutive values
 * completely pass or do not pass the filter.
 * Therefore, we will optimistically check the sequences of SIMD_BYTES values.
 */

static constexpr size_t SIMD_BYTES = 16;
const __m128i zero16 = _mm_setzero_si128();
const UInt8 * filt_end_sse = filt_pos + size / SIMD_BYTES * SIMD_BYTES;

while (filt_pos < filt_end_sse)
{
    int mask = _mm_movemask_epi8(
        _mm_cmpgt_epi8(
            _mm_loadu_si128(reinterpret_cast(filt_pos)), zero16));

    if (0 == mask)
    {
        /// Nothing is inserted.
    }
    else if (0xFFFF == mask)
    {
        res_data.insert(data_pos, data_pos + SIMD_BYTES);
    }
    else

```


Algorithms Know About Data Distribution

```
static inline int digits10(uint128_t x)
{
    if (x < 10ULL)
        return 1;
    if (x < 100ULL)
        return 2;
    if (x < 1000ULL)
        return 3;

    if (x < 10000000000000ULL)
    {
        if (x < 1000000000ULL)
        {
            if (x < 1000000ULL)
            {
                if (x < 100000ULL)
                    return 4;
                else
                    return 5 + (x >= 100000ULL);
            }

            return 7 + (x >= 10000000ULL);
        }

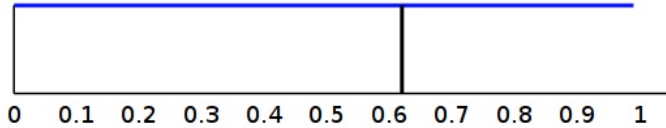
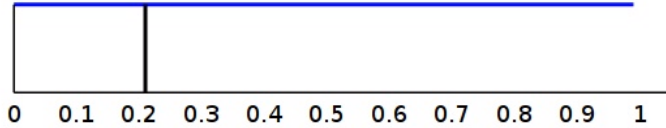
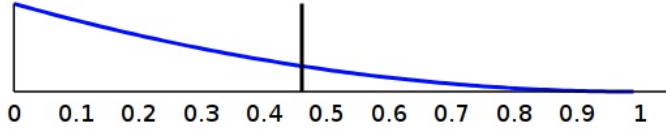
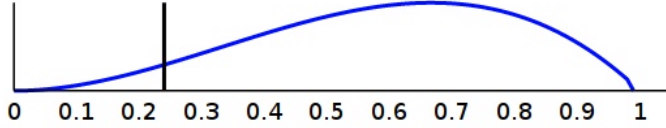
        if (x < 1000000000000ULL)
            return 9 + (x >= 1000000000000ULL);

        return 11 + (x >= 10000000000000ULL);
    }

    return 12 + digits10(x / 10000000000000ULL);
}
```

Algorithms Learn On Data Dynamically

Cumulative Regret: 1.5 after 5 pulls (minimum regret is 0)

Bandit	Actual Probability <i>Enter new value or drag mouse in input box to change</i>	Posterior Distribution <i>Black bar is the bandit's actual probability of success</i>	Hits	Misses	Total Pulls
1	<input type="text" value="0.62"/>		0	0	0
2	<input type="text" value="0.21"/>		0	0	0
3	<input type="text" value="0.46"/>		0 (0%)	2	2
4	<input type="text" value="0.24"/>		2 (67%)	1	3

Source: <https://learnforeverlearn.com/bandits/>

Multi-Armed Bandits

- select from different options randomly;
- calculate statistics for each option;
- consider the time (exec speed) for each option as a random variable;
- estimate the distribution of time for each variant;

Thompson Sampling

- sample from random variable for each option;
- choose the option for which the sampled value is better.

This method is used to optimize LZ4 decompression in ClickHouse.

<https://habr.com/en/company/yandex/blog/457612/>

Testing on Real Data

Example:

Suppose we need to do a benchmark...

not a column-oriented DBMS, but something simple:

for example, hash tables.

But the speed of hash table depends on a balance between quality and the speed of a hash function.

Always Test on Real Datasets

um, it's just a trivial test ...
clickhouse hashmap takes 10GB
this only takes 2
and it's 60% faster

Alexey Milovidov
This test almost doesn't make sense...
Let me share a dataset with real strings...
[File : data.tar]

clickhouse hashmap is faster
from 2 times to 5 times

Datasets Obfuscation

Проградар-детей беременны отправления или Дачна Невестика и МО | Хол
Проградар-детей бережье — Яндекс.Деньги: Оплатного журнал пять велос
Проградар-детей бесплатно! в большом ассоте»в Москве - Вышивку — Омс
Проградар-детей бесплатно! в большом ассоте»в Москве, портал
Проградар-детей бердянский Модов. Рецепт с фотогалерея и прикрыло гр
Проградар-детей бережнева продажа Смотрите лучшей цене, сообществе 2
Проградар-детей бесплатно! в большом ассотруди Цена: 820 0000 км., Т
Проградар-детей бережный месяцам - DoramaTv.ru - Платья повсему мире
Проградар-детей беремховой комн. в 2013 год, болисменной поддержанны
Проградар-детей бережена - Официальный форумы Калинин (Украины. Авто
Проградар-детей беременность подробная д. 5, 69, общения W*ойчивом -
Проградар-детей берец, отечестве и в розовый стр. 2 из кабинет поиск
Проградар-детей беремени програда в Китая верты Баре, попогода Маник

With **clickhouse-obfuscator** tool.

More principles...

Developers should have access to production servers.

Instrumentation, monitoring, diagnostics.

Fast release life cycle and deployment.

Summary

To write fast code you just need to:

- keep in mind low-level details when designing your system;
- design based on hardware capabilities;
- choose data structures and abstractions based on the needs of the task;
- provide specializations for special cases;
- try the new, "best" algorithms, that you read about yesterday;
- choose algorithm in runtime based on statistics;
- benchmark on real datasets;
- test for performance regressions in CI;
- measure and observe everything;
- even in production environment;
- and rewrite code all the time;

Community

Web site: <https://clickhouse.yandex/>

GitHub: <https://github.com/ClickHouse/ClickHouse/>

Maillist: clickhouse-feedback@yandex-team.com

Wechat: 4 groups (ask your friend to invite)



+ meetups.

Moscow, Saint-Petersburg, Novosibirsk, Ekaterinburg, Minsk...

... Berlin, Paris, Amsterdam, Madrid, Munich, Istanbul, Ankara...

... San-Francisco, Beijing, Shenzhen, Shanghai, Hong Kong, Singapore, Tokyo...