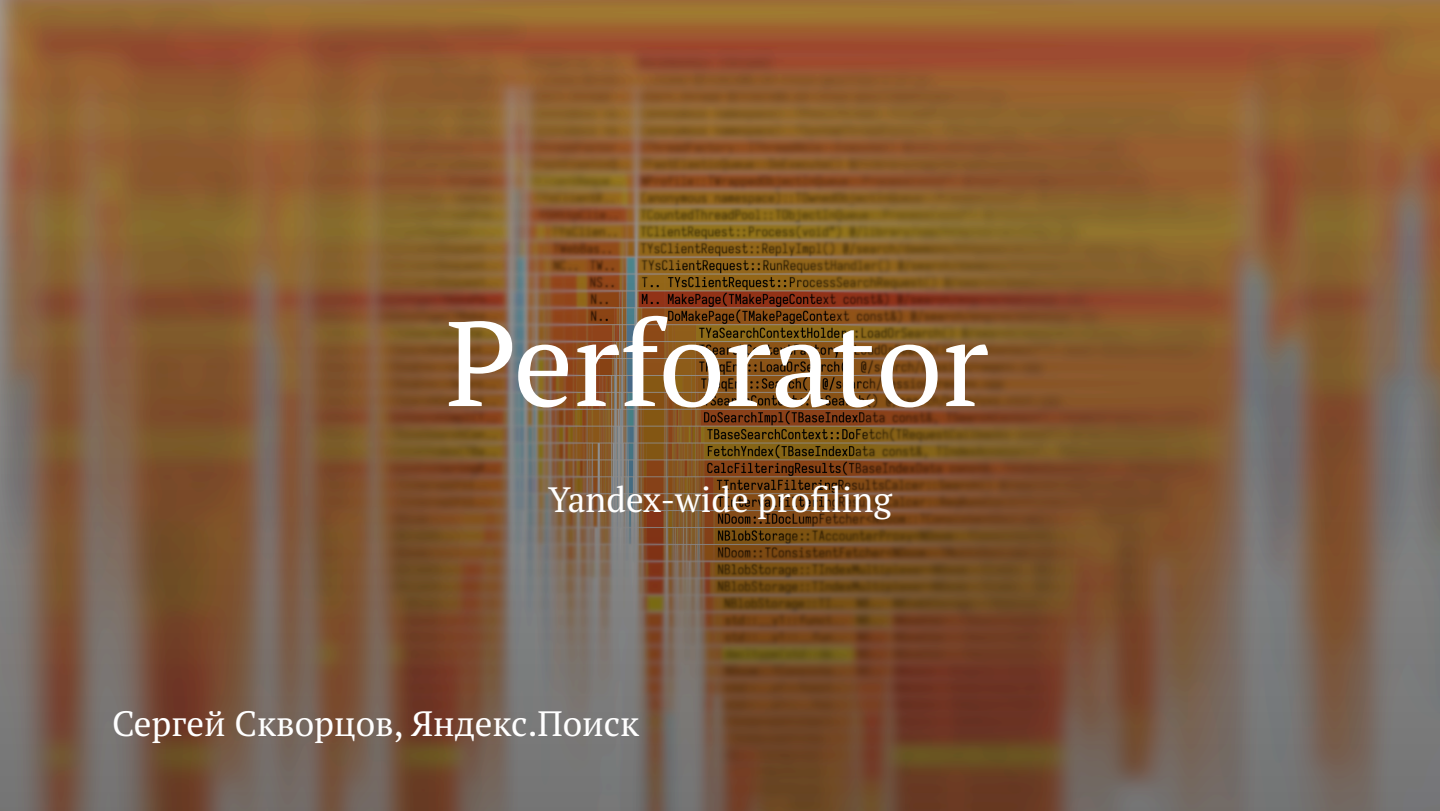


Perforator

Yandex-wide profiling

Сергей Скворцов, Яндекс.Поиск

Perforator

Примерный план

- Обсудим известные проблемы профилирования
  - Вспомним, как работает perf
  - Изучим подходы к раскрутке стека
- Посмотрим, как построить cluster-wide профилировщик
  - Локальный профилировщик
  - Распределенный бэкенд
- Придумаем, как использовать полученную конструкцию

Perforator

Предметная область

- Linux
- x86-64, AArch64
- Нативные языки: C++, C, Go, Rust

Как профилировать?

Профилировать на масштабах одной программы мы умеем

Как профилировать?

Профилировать на масштабах одной программы мы умеем

- GDB: https://poormansprofiler.org

Как профилировать?

Профилировать на масштабах одной программы мы умеем

- GDB: https://poormansprofiler.org
- Linux perf

Как профилировать?

Профилировать на масштабах одной программы мы умеем

- GDB: https://poormansprofiler.org
- Linux perf
- Intel VTune

Как профилировать?

Профилировать на масштабах одной программы мы умеем

- GDB: https://poormansprofiler.org
- Linux perf
- Intel VTune
- Интегрированные профилировщики

Как профилировать?

Профилировать на масштабах одной программы мы умеем

- GDB: https://poormansprofiler.org
- Linux perf
- Intel VTune
- Интегрированные профилировщики
- …

Как профилировать весь Яндекс?

Как профилировать весь Яндекс?

Нам нужен профилировщик для всего кластера

- Где сервисы тратят железо?
- Как оценивать результат оптимизаций общих компонент?
- Как ведет себя конкретный сервер?
- Как устроен конкретный сервис?
- Что общего в профиле нагрузки у разных сервисов?

Как профилировать весь Яндекс?

Множество гетерогенных сервисов

- Разные языки
- Разные версии рантайма и компиляторов
- Разные режимы сборки

Виды профилировщиков

Выбираем профилировщик

- Инструментирующие
- Семплирующие

Linux perf

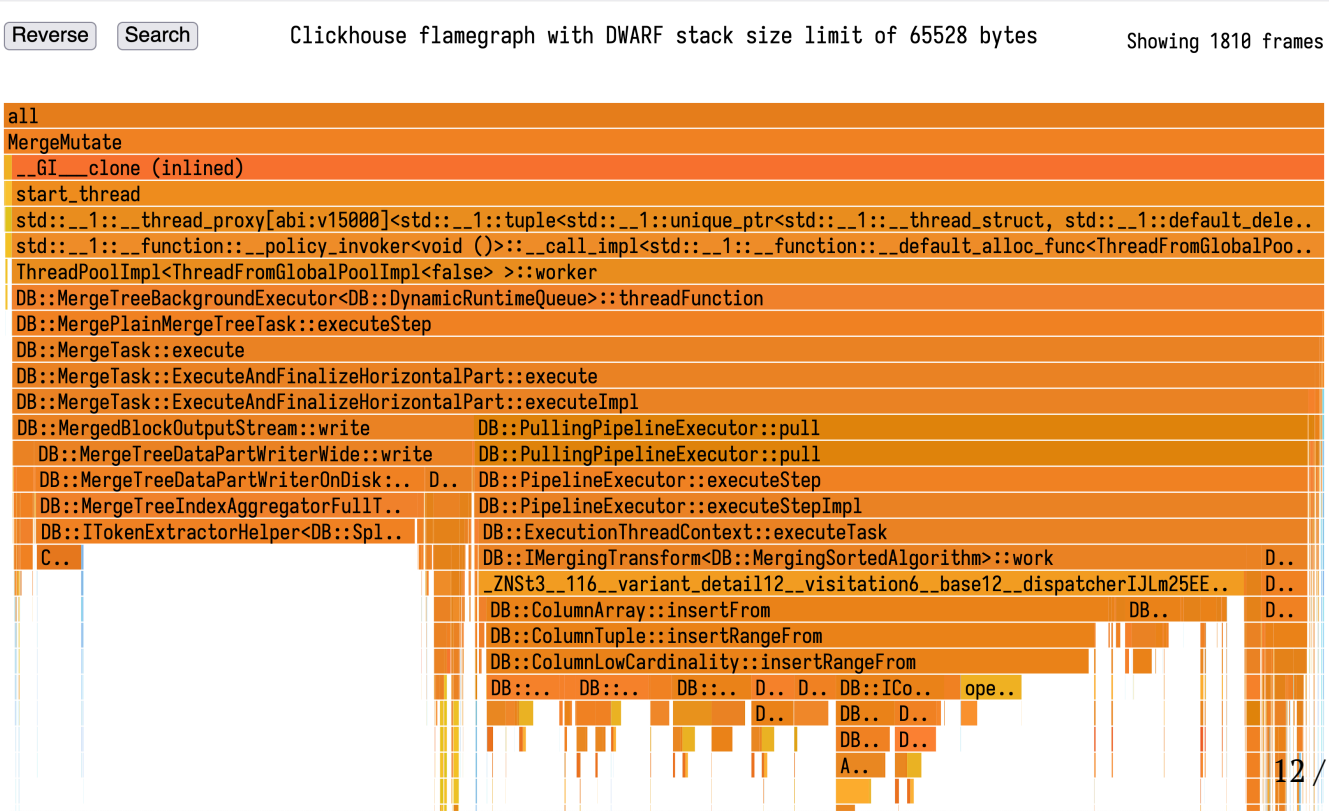
- Стандартный профилировщик под Linux
- Много режимов работы, нам интересен семплирующий `perf record`
- Поддерживает много хардварных и софтварных счетчиков
- Работает через Performance Monitoring Unit (PMU)

Performance Monitoring Unit (PMU)

Технология для анализа производительности CPU

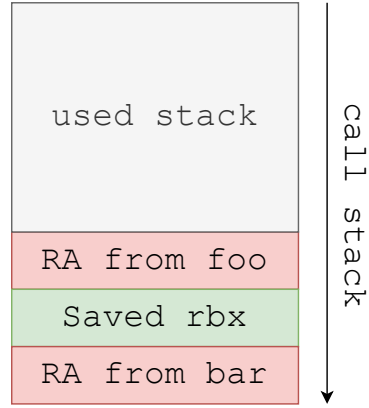
- CPU раз в N событий вызывает особое прерывание
- Ядро Linux его обрабатывает и анализирует состояние процесса
- Нельзя обработать в userspace
- Разные типы событий: такты, инструкции, промахи мимо кеша и сотни специфичных для архитектуры

```
perf record -e cycles -c 1000000 -p 12345
```


Reverse
Search
Clickhouse flamegraph with DWARF stack size limit of 65528 bytes
Showing 1810 frames
all
MergeMutate
__GI___clone (inlined)
start_thread
std::__1::__thread_proxy[abi:v15000]<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct, std::__1::default_dele..
std::__1::__function::__policy_invoker<void ()>::__call_impl<std::__1::__function::__default_alloc_func<ThreadFromGlobalPoo..
ThreadPoolImpl<ThreadFromGlobalPoolImpl<false> >::worker
DB::MergeTreeBackgroundExecutor<DB::DynamicRuntimeQueue>::threadFunction
DB::MergePlainMergeTreeTask::executeStep
DB::MergeTask::execute
DB::MergeTask::ExecuteAndFinalizeHorizontalPart::execute
DB::MergeTask::ExecuteAndFinalizeHorizontalPart::executeImpl
DB::MergedBlockOutputStream::write
DB::PullingPipelineExecutor::pull
DB::MergeTreeDataPartWriterWide::write
DB::PullingPipelineExecutor::pull
DB::MergeTreeDataPartWriterOnDisk::..
D..
DB::PipelineExecutor::executeStep
DB::MergeTreeIndexAggregatorFullT..
DB::PipelineExecutor::executeStepImpl
DB::ITokenExtractorHelper<DB::Spl..
DB::ExecutionThreadContext::executeTask
C..
DB::IMergingTransform<DB::MergingSortedAlgorithm>::work
D..
_ZNSt3__116__variant_detail12__visitation6__base12__dispatcherIJLm25EE..
D..
DB::ColumnArray::insertFrom
DB..
D..
DB::ColumnTuple::insertRangeFrom
DB::ColumnLowCardinality::insertRangeFrom
DB::..
DB::..
DB::..
D..
D..
DB::ICo..
ope..
D..
DB..
D..
DB..
D..
A..

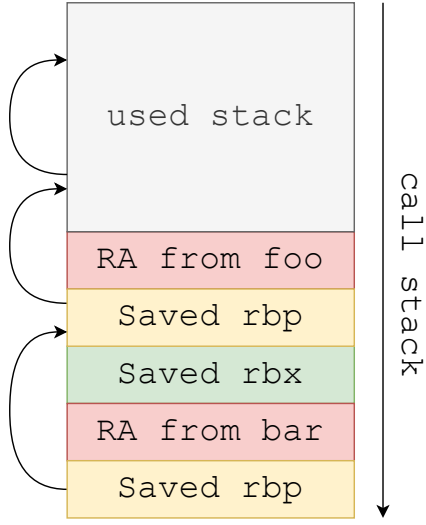
Раскрутка стека: frame pointers

```
foo:
        pushq   %rbx
        movl    %edx, %ebx
        callq   bar
        movl    %eax, %edi
        movl    %ebx, %esi
        callq   bar
        popq    %rbx
        retq
```



Раскрутка стека: frame pointers

```
foo:
        pushq   %rbp        ; prologue
        movq    %rsp, %rbp  ; prologue
        pushq   %rbx
        movl    %edx, %ebx
        callq   bar
        movl    %eax, %edi
        movl    %ebx, %esi
        callq   bar
        popq    %rbx
        popq    %rbp        ; epilogue
        retq
```



Раскрутка стека: frame pointers

- `perf` прямо в ядре Linux собирает стек вызовов
- Код раскрутки стека очень простой, легко верифицировать

Что не так с frame pointers?

Почему бы не включить везде?

Что не так с frame pointers?

Почему бы не включить везде?

- Теряем производительность: типично 1-2%

Что не так с frame pointers?

Почему бы не включить везде?

- Теряем производительность: типично 1-2%
- Просадка сильно зависит от нагрузки: **в Clickhouse до 40%**

Что не так с frame pointers?

Почему бы не включить везде?

- Теряем производительность: типично 1-2%
- Просадка сильно зависит от нагрузки: **в Clickhouse до 40%**
- Нужно пересобирать весь мир: **долго сходится**

Что не так с frame pointers?

Почему бы не включить везде?

- Теряем производительность: типично 1-2%
- Просадка сильно зависит от нагрузки: **в Clickhouse до 40%**
- Нужно пересобирать весь мир: **долго сходится**
- Есть библиотеки и приложения, которые **не удастся пересобрать**

Что не так с frame pointers?

Почему бы не включить везде?

- Теряем производительность: типично 1-2%
- Просадка сильно зависит от нагрузки: **в Clickhouse до 40%**
- Нужно пересобирать весь мир: **долго сходится**
- Есть библиотеки и приложения, которые **не удастся пересобрать**
- Есть рукописный ассемблер: нужно **править много кода**

Раскрутка стека: DWARF

DWARF – формат отладочной информации

Для каждой инструкции в исполняемом файле закодировано, как раскрутить стек из неё

- Нужно вычислить состояние регистров на момент вызова функции
- Из `rsp` тривиально получается адрес возврата
- Повторяем для следующей функции

Раскрутка стека: DWARF

Правила для вычисления состояния регистров в начале функции

```
1   foo:
2     push   %rbp
3     push   %r14
4     push   %rbx
5     sub    $0x400,%rsp
6     ...
7     add    $0x400,%rsp
8     pop    %rbx
9     pop    %r14
10    pop    %rbp
11    ret
```

```
1   CFA      RA
2   rsp+8    CFA-8
3   rsp+16   CFA-8
4   rsp+24   CFA-8
5   rsp+32   CFA-8
6   rsp+1056 CFA-8
7   ...
8   rsp+32   CFA-8
9   rsp+24   CFA-8
10  rsp+16   CFA-8
11  rsp+8    CFA-8
```

Раскрутка стека: DWARF

Правила для вычисления состояния регистров в начале функции

```
1   foo:
2     push   %rbp
3     mov    %rsp, %rbp
4     push   %r14
5     push   %rbx
6     sub    $0x400,%rsp
7     ...
8     add    $0x400,%rsp
9     pop    %rbx
10    pop    %r14
11    pop    %rbp
12    ret
```

```
1   CFA       RA
2   rsp+8     CFA-8
3   rbp+16    CFA-8
4   rbp+16    CFA-8
5   rbp+16    CFA-8
6   rbp+16    CFA-8
7   rbp+16    CFA-8
8   ...
9   rbp+16    CFA-8
10  rbp+16    CFA-8
11  rbp+16    CFA-8
12  rsp+8     CFA-8
```

```
CFA = *(rsp + 8 + r9 << 3) + 8
```

DWARF не раскрутить в ядре Linux

Ответ на попытку реализовать раскрутку через DWARF:

DWARF не раскрутить в ядре Linux

Ответ на попытку реализовать раскрутку через DWARF:

> I never *ever* want to see this code ever again. (…) An unwinder that is several hundred lines long is simply not even *remotely* interesting to me. (…) In the absence of that, just follow the damn chain on the stack *without* the "smarts" of an inevitably buggy piece of crap.

– Linus Torvalds

Что не так с DWARF?

Приходится раскручивать стек в userspace

- Копирует верхушку стека из kernelspace в userspace на каждый семпл
- Постпроцессинг: парсит DWARF и раскручивает стек уже после записи
- Большой размер трассы: **сотни мегабайт в секунду**
- Обрезает стеки: максимум 64KiB из 2MiB доступного стека

Почему не perf?

Есть фатальный недостаток

Почему не perf?

Есть фатальный недостаток

Нужно пересобирать программы

```
1   $ perf record --call-graph=fp
```

Почему не perf?

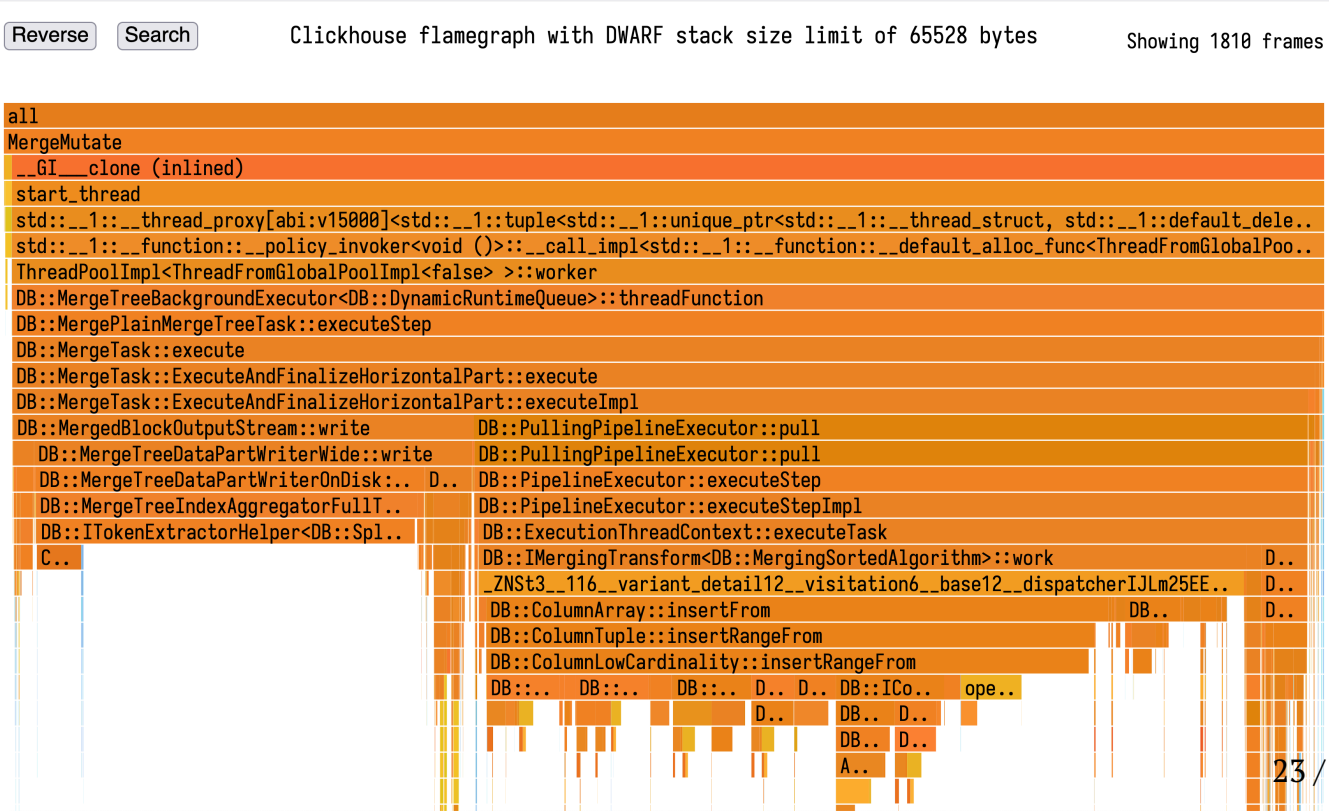
Есть фатальный недостаток

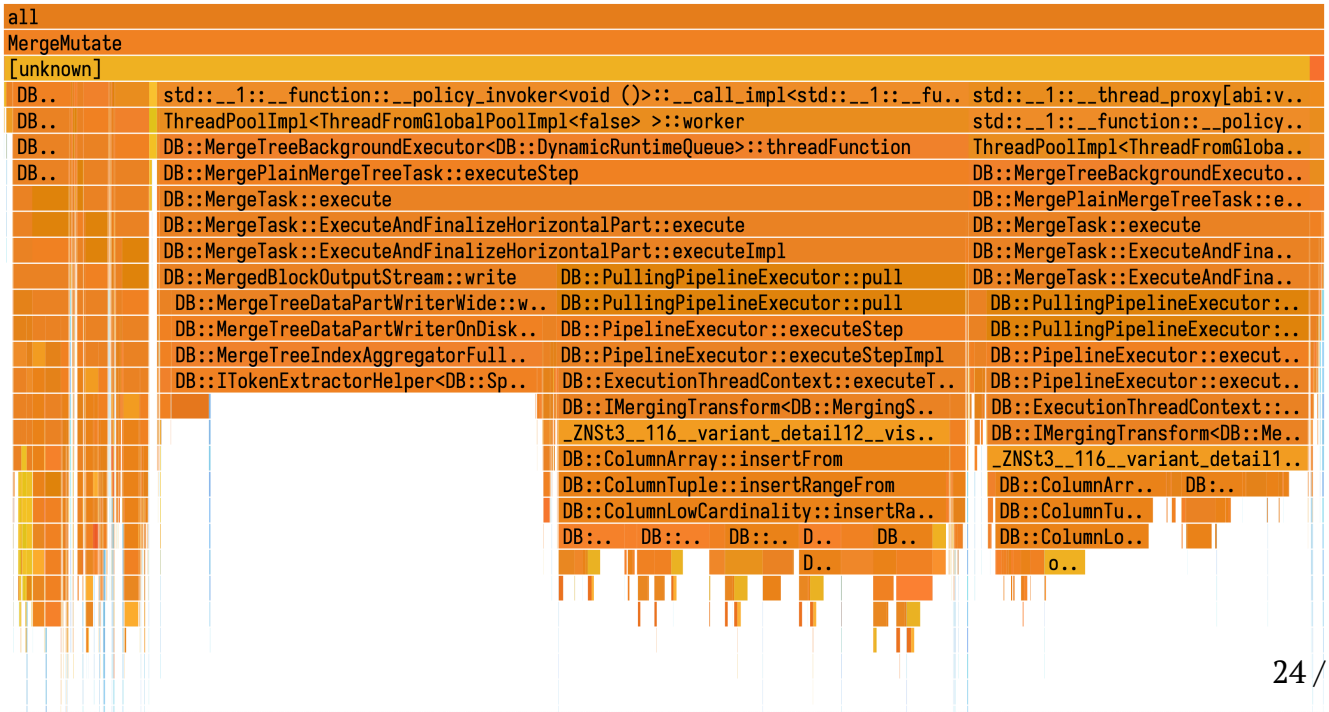
Нужно пересобирать программы

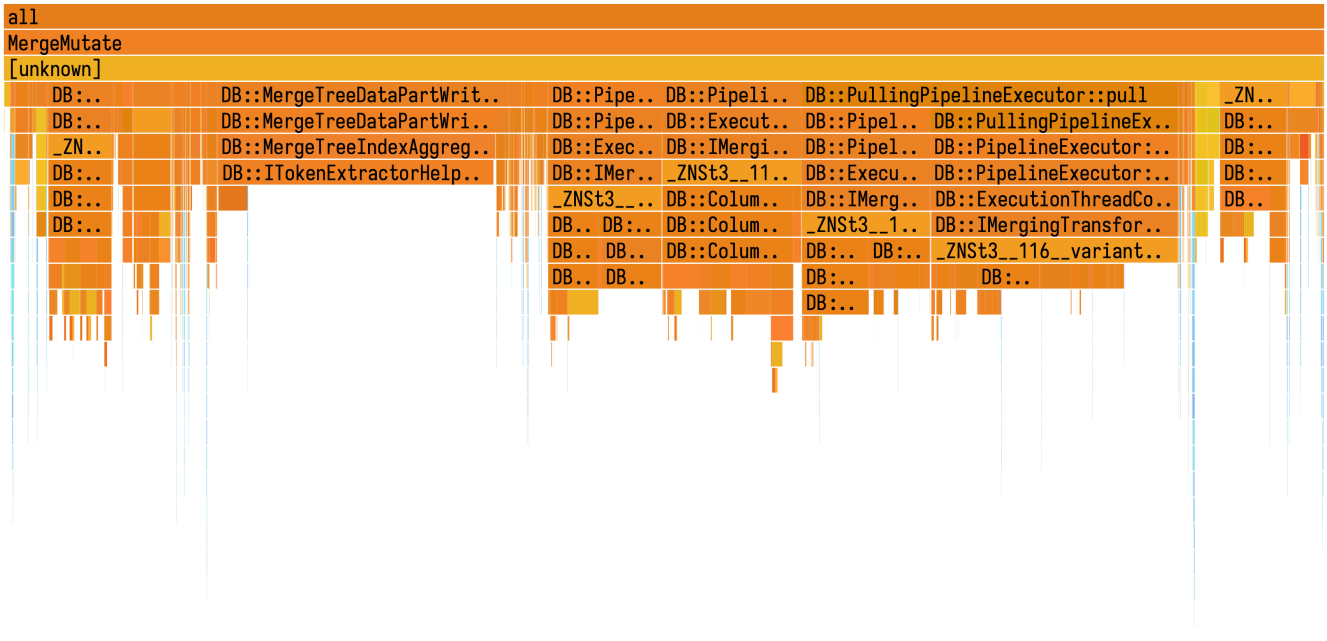
```
1   $ perf record --call-graph=fp
```

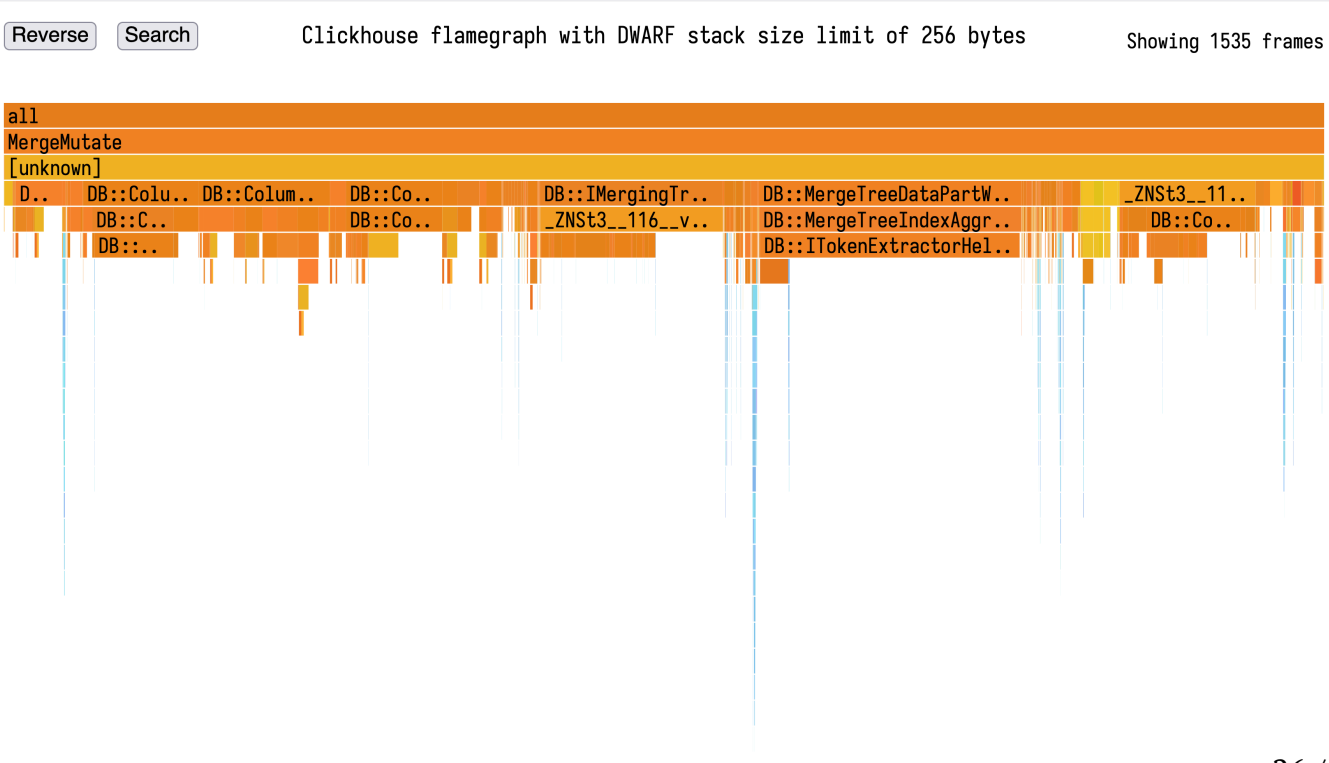
Обрезанные стеки

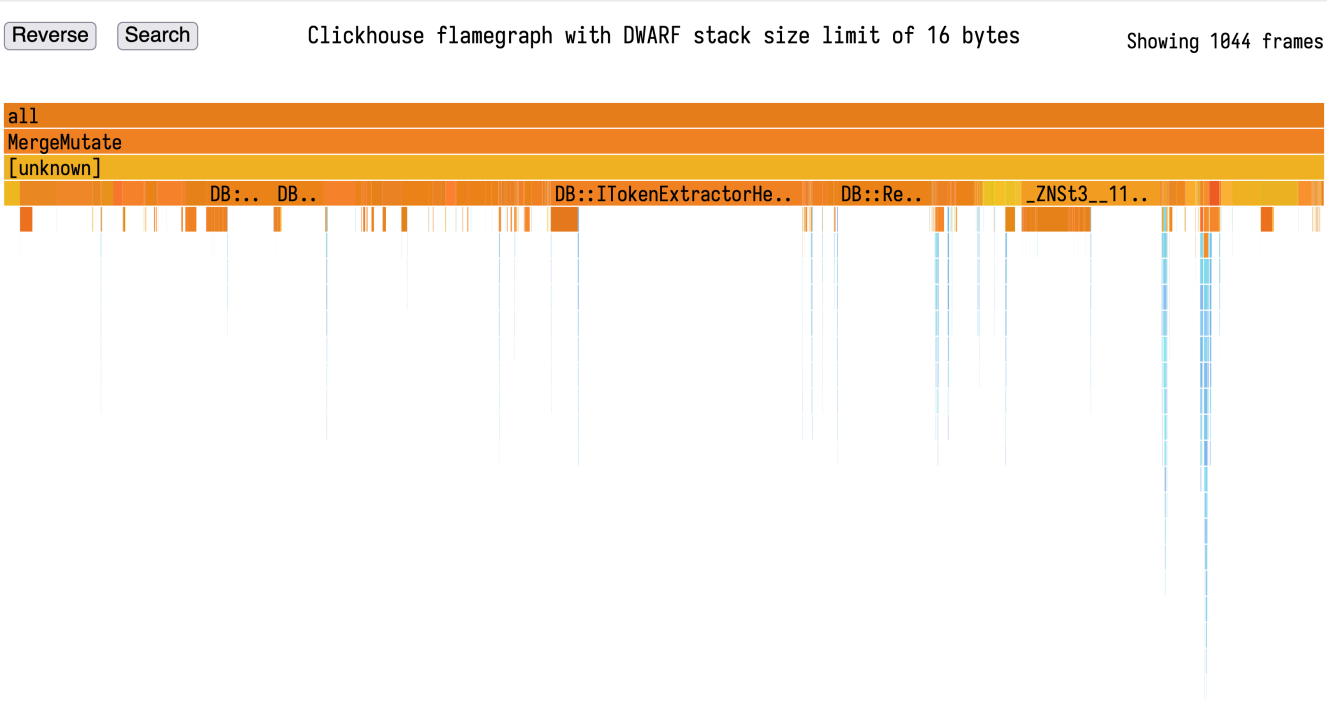
```
1   $ perf record --call-graph=dwarf
```

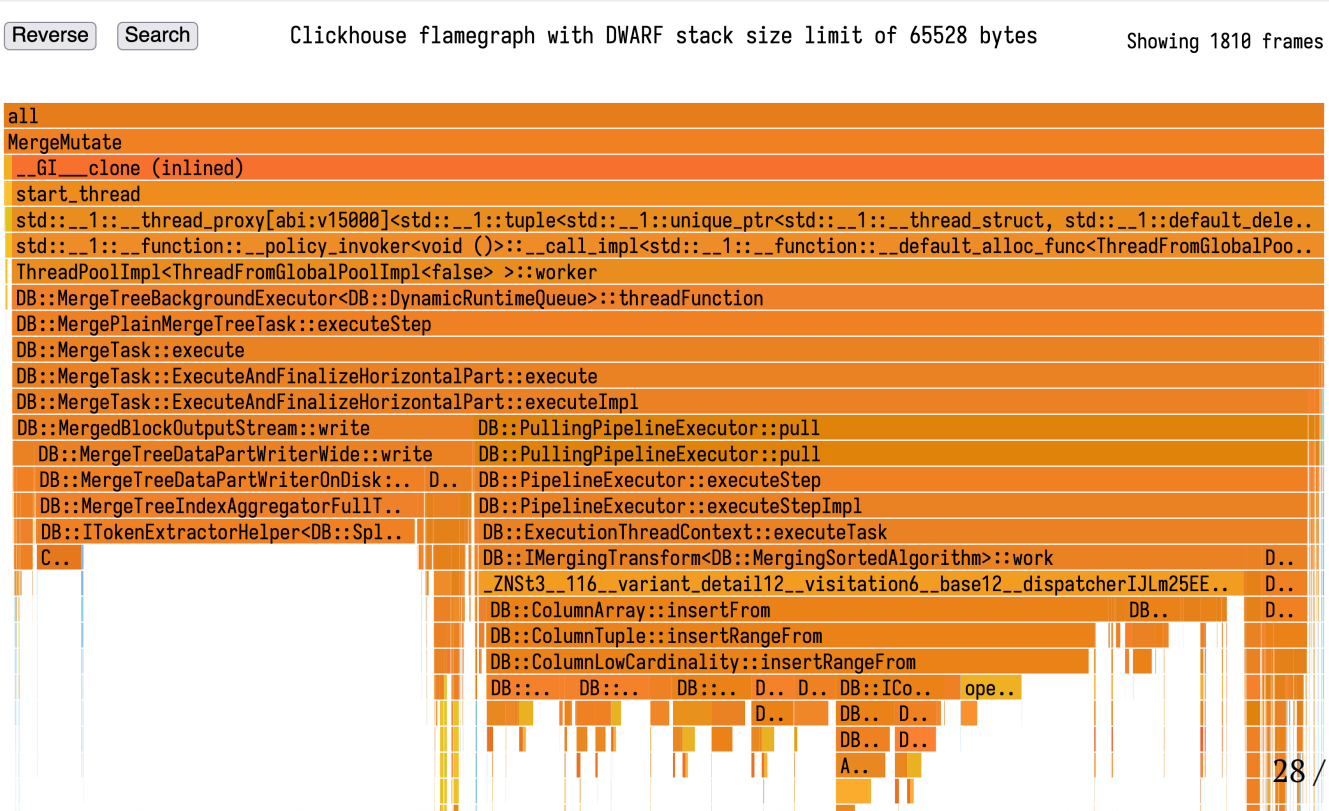

Reverse
Search
Clickhouse flamegraph with DWARF stack size limit of 65528 bytes
Showing 1810 frames
all
MergeMutate
__GI___clone (inlined)
start_thread
std::__1::__thread_proxy[abi:v15000]<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct, std::__1::default_dele..
std::__1::__function::__policy_invoker<void ()>::__call_impl<std::__1::__function::__default_alloc_func<ThreadFromGlobalPoo..
ThreadPoolImpl<ThreadFromGlobalPoolImpl<false> >::worker
DB::MergeTreeBackgroundExecutor<DB::DynamicRuntimeQueue>::threadFunction
DB::MergePlainMergeTreeTask::executeStep
DB::MergeTask::execute
DB::MergeTask::ExecuteAndFinalizeHorizontalPart::execute
DB::MergeTask::ExecuteAndFinalizeHorizontalPart::executeImpl
DB::MergedBlockOutputStream::write
DB::PullingPipelineExecutor::pull
DB::MergeTreeDataPartWriterWide::write
DB::PullingPipelineExecutor::pull
DB::MergeTreeDataPartWriterOnDisk::..
D..
DB::PipelineExecutor::executeStep
DB::MergeTreeIndexAggregatorFullT..
DB::PipelineExecutor::executeStepImpl
DB::ITokenExtractorHelper<DB::Spl..
DB::ExecutionThreadContext::executeTask
C..
DB::IMergingTransform<DB::MergingSortedAlgorithm>::work
D..
_ZNSt3__116__variant_detail12__visitation6__base12__dispatcherIJLm25EE..
D..
DB::ColumnArray::insertFrom
DB..
D..
DB::ColumnTuple::insertRangeFrom
DB::ColumnLowCardinality::insertRangeFrom
DB::..
DB::..
DB::..
D..
D..
DB::ICo..
ope..
D..
DB..
D..
DB..
D..
A..


Reverse
Search
Clickhouse flamegraph with DWARF stack size limit of 4096 bytes
Showing 2807 frames
all
MergeMutate
[unknown]
DB..
std::__1::__function::__policy_invoker<void ()>::__call_impl<std::__1::__fu..
std::__1::__thread_proxy[abi:v..
ThreadPoolImpl<ThreadFromGlobalPoolImpl<false> >::worker
std::__1::__function::__policy..
DB::MergeTreeBackgroundExecutor<DB::DynamicRuntimeQueue>::threadFunction
ThreadPoolImpl<ThreadFromGloba..
DB::MergePlainMergeTreeTask::executeStep
DB::MergeTreeBackgroundExecuto..
DB::MergeTask::execute
DB::MergePlainMergeTreeTask::e..
DB::MergeTask::ExecuteAndFinalizeHorizontalPart::execute
DB::MergeTask::execute
DB::MergeTask::ExecuteAndFinalizeHorizontalPart::executeImpl
DB::MergeTask::ExecuteAndFina..
DB::MergedBlockOutputStream::write
DB::PullingPipelineExecutor::pull
DB::MergeTask::ExecuteAndFina..
DB::MergeTreeDataPartWriterWide::w..
DB::PullingPipelineExecutor::pull
DB::PullingPipelineExecutor::..
DB::MergeTreeDataPartWriterOnDisk..
DB::PipelineExecutor::executeStep
DB::PullingPipelineExecutor::..
DB::MergeTreeIndexAggregatorFull..
DB::PipelineExecutor::executeStepImpl
DB::PipelineExecutor::execut..
DB::ITokenExtractorHelper<DB::Sp..
DB::ExecutionThreadContext::executeT..
DB::PipelineExecutor::execut..
DB::IMergingTransform<DB::MergingS..
DB::ExecutionThreadContext::..
_ZNSt3__116__variant_detail12__vis..
DB::IMergingTransform<DB::Me..
DB::ColumnArray::insertFrom
_ZNSt3__116__variant_detail1..
DB::ColumnTuple::insertRangeFrom
DB::ColumnArr..
DB::..
DB::ColumnLowCardinality::insertRa..
DB::ColumnTu..
DB::..
DB::..
DB::..
D..
DB..
DB::ColumnLo..
D..
o..

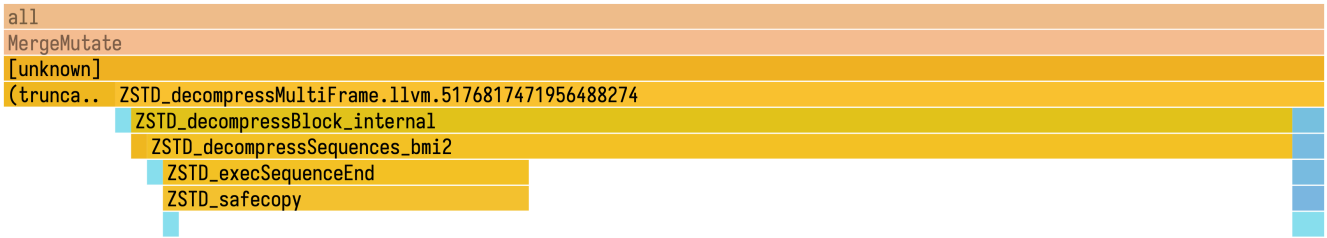

Reverse
Search
Clickhouse flamegraph with DWARF stack size limit of 1024 bytes
Showing 1925 frames
all
MergeMutate
[unknown]
DB::..
DB::MergeTreeDataPartWrit..
DB::Pipe..
DB::Pipeli..
DB::PullingPipelineExecutor::pull
_ZN..
DB::..
DB::MergeTreeDataPartWri..
DB::Pipe..
DB::Execut..
DB::Pipel..
DB::PullingPipelineEx..
DB::..
_ZN..
DB::MergeTreeIndexAggreg..
DB::Exec..
DB::IMergi..
DB::Pipel..
DB::PipelineExecutor::..
DB::..
DB::..
DB::ITokenExtractorHelp..
DB::IMer..
_ZNSt3__11..
DB::Execu..
DB::PipelineExecutor::..
DB::..
DB::..
_ZNSt3__..
DB::Colum..
DB::IMerg..
DB::ExecutionThreadCo..
DB..
DB::..
DB:..
DB::Colum..
_ZNSt3__1..
DB::IMergingTransfor..
DB..
DB..
DB::Colum..
DB::..
DB::..
_ZNSt3__116__variant..
DB..
DB..
DB::..
DB::..
DB::..


Reverse
Search
Clickhouse flamegraph with DWARF stack size limit of 256 bytes
Showing 1535 frames
all
MergeMutate
[unknown]
D..
DB::Colu..
DB::Colum..
DB::Co..
DB::IMergingTr..
DB::MergeTreeDataPartW..
_ZNSt3__11..
DB::C..
DB::Co..
_ZNSt3__116__v..
DB::MergeTreeIndexAggr..
DB::Co..
DB::..
DB::ITokenExtractorHel..


Reverse
Search
Clickhouse flamegraph with DWARF stack size limit of 16 bytes
Showing 1044 frames
all
MergeMutate
[unknown]
DB:..
DB..
DB::ITokenExtractorHe..
DB::Re..
_ZNSt3__11..

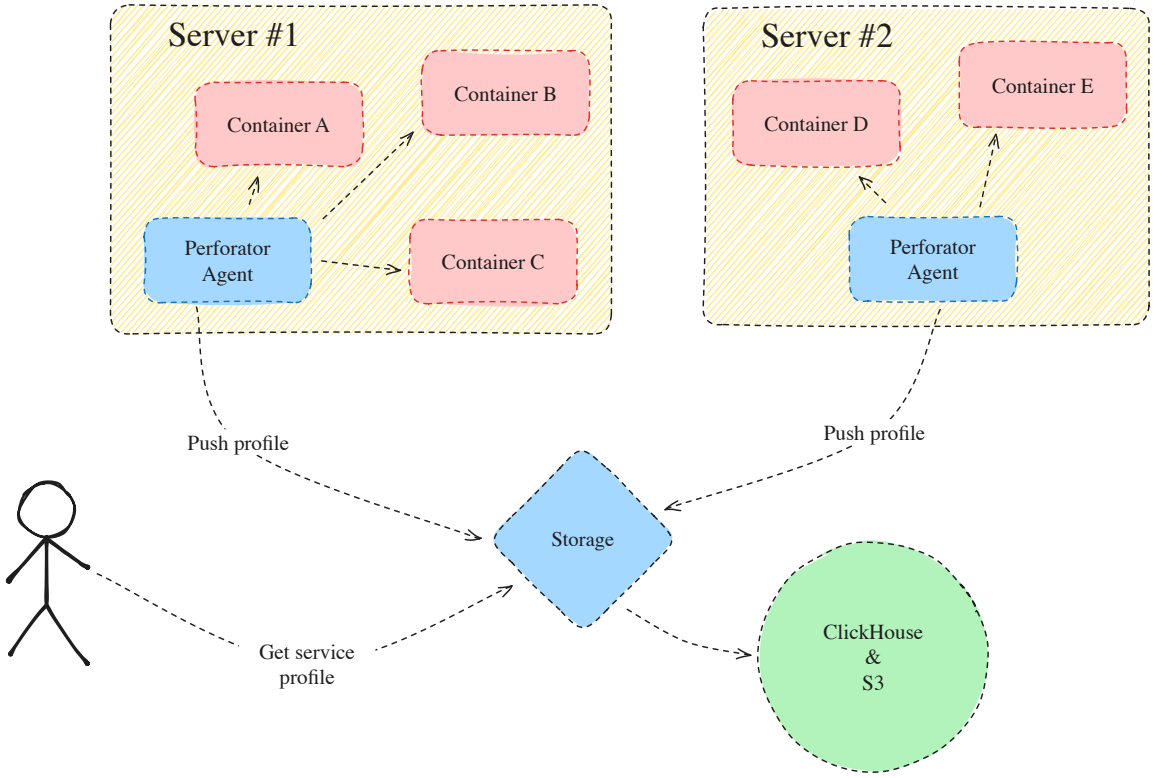

Reverse
Search
Clickhouse flamegraph with DWARF stack size limit of 65528 bytes
Showing 1810 frames
all
MergeMutate
__GI___clone (inlined)
start_thread
std::__1::__thread_proxy[abi:v15000]<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct, std::__1::default_dele..
std::__1::__function::__policy_invoker<void ()>::__call_impl<std::__1::__function::__default_alloc_func<ThreadFromGlobalPoo..
ThreadPoolImpl<ThreadFromGlobalPoolImpl<false> >::worker
DB::MergeTreeBackgroundExecutor<DB::DynamicRuntimeQueue>::threadFunction
DB::MergePlainMergeTreeTask::executeStep
DB::MergeTask::execute
DB::MergeTask::ExecuteAndFinalizeHorizontalPart::execute
DB::MergeTask::ExecuteAndFinalizeHorizontalPart::executeImpl
DB::MergedBlockOutputStream::write
DB::PullingPipelineExecutor::pull
DB::MergeTreeDataPartWriterWide::write
DB::PullingPipelineExecutor::pull
DB::MergeTreeDataPartWriterOnDisk::..
D..
DB::PipelineExecutor::executeStep
DB::MergeTreeIndexAggregatorFullT..
DB::PipelineExecutor::executeStepImpl
DB::ITokenExtractorHelper<DB::Spl..
DB::ExecutionThreadContext::executeTask
C..
DB::IMergingTransform<DB::MergingSortedAlgorithm>::work
D..
_ZNSt3__116__variant_detail12__visitation6__base12__dispatcherIJLm25EE..
D..
DB::ColumnArray::insertFrom
DB..
D..
DB::ColumnTuple::insertRangeFrom
DB::ColumnLowCardinality::insertRangeFrom
DB::..
DB::..
DB::..
D..
D..
DB::ICo..
ope..
D..
DB..
D..
DB..
D..
A..


Reverse
Search
Clickhouse flamegraph with DWARF stack size limit of 65528 bytes
Showing 1810 frames
all
MergeMutate
__GI___clone (inlined)
start_thread
std::__1::__thread_proxy[abi:v15000]<std::__1::tuple<std::__1::unique_ptr<std::__1::__thread_struct, std::__1::default_dele..
std::__1::__function::__policy_invoker<void ()>::__call_impl<std::__1::__function::__default_alloc_func<ThreadFromGlobalPoo..
ThreadPoolImpl<ThreadFromGlobalPoolImpl<false> >::worker
DB::MergeTreeBackgroundExecutor<DB::DynamicRuntimeQueue>::threadFunction
DB::MergePlainMergeTreeTask::executeStep
DB::MergeTask::execute
DB::MergeTask::ExecuteAndFinalizeHorizontalPart::execute
DB::MergeTask::ExecuteAndFinalizeHorizontalPart::executeImpl
DB::MergedBlockOutputStream::write
DB::MergeTreeDataPartWideWide::write
DB::MergeTreeDataPartWriterOnDisk::...
DB::MergeTreeIndexAggregatorFullT..
DB::ITokenExtractorHelper<DB::Spl...
C..
DB::PullingPipelineExecutor::pull
DB::PullingPipelineExecutor::pull
DB::PipelineExecutor::executeStep
DB::PipelineExecutor::executeStepImpl
DB::ExecutionThreadContext::executeTask
DB::IMergingTransform<DB::MergingSortedAlgorithm>::work
_ZNSt3__116__variant_detail12__visitation6__base12__dispatcherIJLm25EE..
DB::ColumnArray::insertFrom
DB::ColumnTuple::insertRangeFrom
DB::ColumnLowCardinality::insertRangeFrom
DB::ICo..
ope..
A..


Reverse
Search
Clickhouse flamegraph with DWARF stack size limit of 65528 bytes
Showing 1810 frames
all
MergeMutate
[unknown]
(trunca..
ZSTD_decompressMultiFrame.llvm.517681747195648827 4
ZSTD_decompressBlock_internal
ZSTD_decompressSequences_bmi2
ZSTD_execSequenceEnd
ZSTD_safecopy

Как быть?


Написать ещё один профилировщик
Включить -fno-omit -frame-pointer


Server #1
Container A
Container B
Perforator Agent
Container C
Server #2
Container D
Container E
Perforator Agent
Push profile
Push profile
Storage
Get service profile
ClickHouse & S3

Perforator agent

- Выполняется на каждой ноде кластера
- Собирает профили со всей ноды и отправляет в базу профилей
- Можно использовать в качестве классического профилировщика
- Не требует модификации кода или пересборки программ

Perforator

Нужно как-то раскрутить стек в ядре Linux

Perforator

Нужно как-то раскрутить стек в ядре Linux

- Патч в ядро

Perforator

Нужно как-то раскрутить стек в ядре Linux

- Патч в ядро
- Модули ядра

Perforator

Нужно как-то раскрутить стек в ядре Linux

- Патч в ядро
- Модули ядра
- eBPF

Perforator

Нужно как-то раскрутить стек в ядре Linux

- Патч в ядро
- Модули ядра
- eBPF

Немного про eBPF

- Верифицируемые программы для специальной VM внутри Linux
- Гарантированно не содержит UB ~~(если только в ядре нет багов)~~

Немного про eBPF

Множество ограничений: не Тьюринг-полные, есть лимит на сложность

- Циклы только на константное число итераций
- Сложно написать бинарный поиск
- Нетривиальные программы нужно долго оптимизировать под верификатор

eBPF meets Linux perf

В Linux есть поддержка вызова eBPF-программы на прерывание от PMU

- Можем построить свой perf
- На каждые N тактов изучаем состояние программы
- Можем понять буквально все про процесс
- Как раскрутить стек?

eBPF meets DWARF?

Если сможем раскрутить через DWARF, то победим. Но как?

eBPF meets DWARF?

Если сможем раскрутить через DWARF, то победим. Но как?

- Нужно выполнить код для **виртуальной машины DWARF**

eBPF meets DWARF?

Если сможем раскрутить через DWARF, то победим. Но как?

- Нужно выполнить код для **виртуальной машины DWARF**
- Посмотрим внимательно на DWARF, который пишут компиляторы

eBPF meets DWARF?

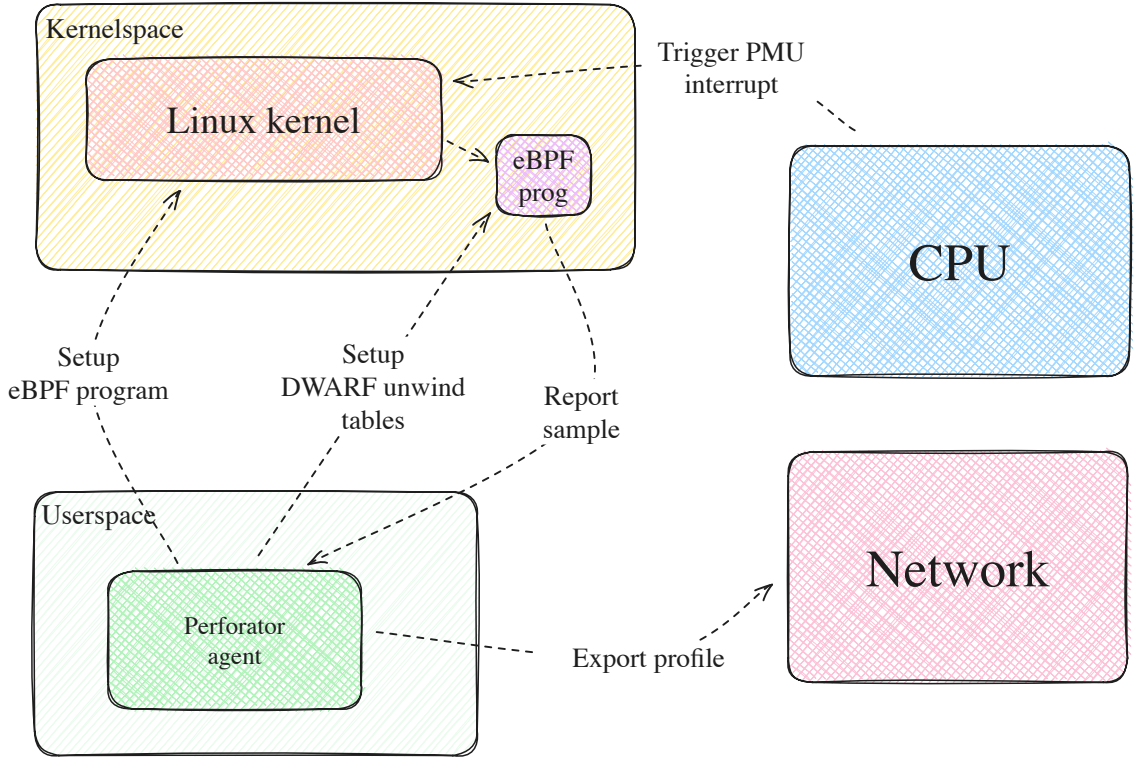
Если сможем раскрутить через DWARF, то победим. Но как?

- Нужно выполнить код для **виртуальной машины DWARF**
- Посмотрим внимательно на DWARF, который пишут компиляторы
- **100% правил раскрутки** – это `rsp+offset` и `rbp+offset`

eBPF meets DWARF?

Если сможем раскрутить через DWARF, то победим. Но как?

- Нужно выполнить код для **виртуальной машины DWARF**
- Посмотрим внимательно на DWARF, который пишут компиляторы
- **100% правил раскрутки** – это `rsp+offset` и `rbp+offset`
- Можно построить облегченную версию таблицы раскрутки без Тьюринг-полноты!


Kernelspace
Linux kernel
eBPF prog
Trigger PMU interrupt
CPU
Setup eBPF program
Setup DWARF unwind tables
Report sample
Userspace
Perforator agent
Export profile
Network

Perforator agent

Про overhead

- CPU: около 0.2% на реальных приложениях
  - Зависит от частоты сбора семплов. У нас 99Hz с одного ядра
- RAM: 1% от памяти хоста
- Сейчас непрерывано профилируем все процессы, без opt-in

Perforator symbolizer

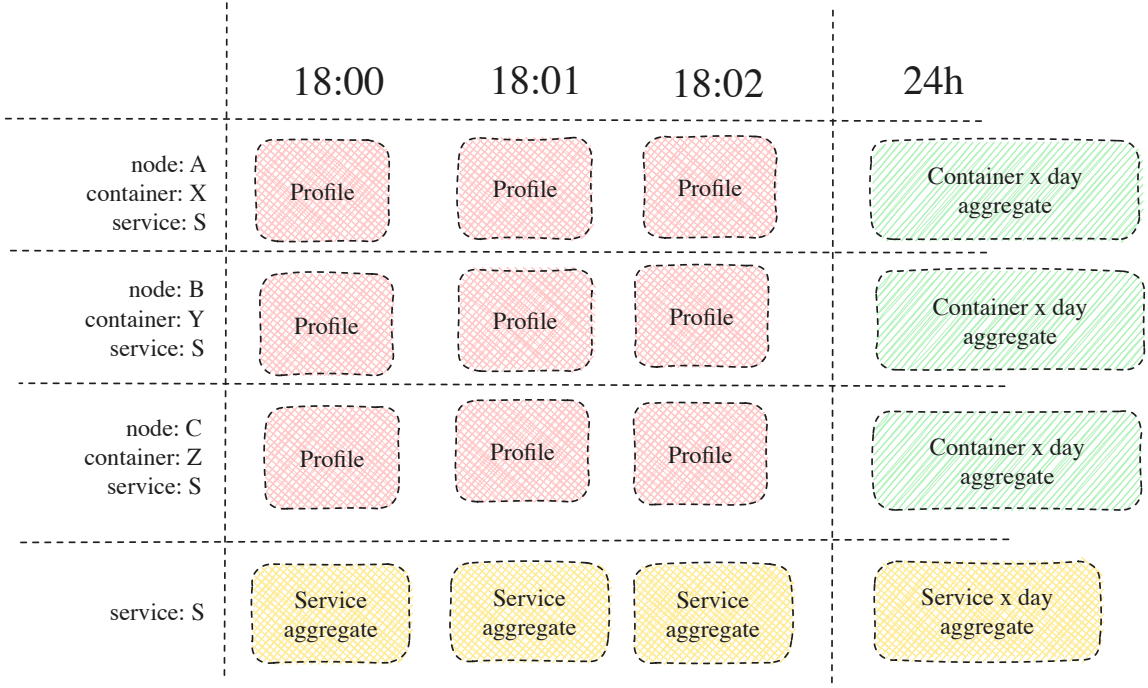
Превращать адреса инструкций в названия функций дорого

- Агенты отправляют адреса и идентификаторы исполняемых файлов
- Символизируем все в одной точке
- Максимально точно показываем заинлайненные функции. Лучше только gdb

Perforator storage

База данных профилей

- Хранилище поверх Clickhouse + S3
- Принимает профили из произвольных источников в формате **pprof**
- Хранит последовательности атомарных профилей. Ключ последовательности – набор меток
- По разным проекциям строит предагрегаты


18:00
18:01
18:02
24h
node: A
container: X
service: S
Profile
Profile
Profile
Container x day aggregate
node: B
container: Y
service: S
Profile
Profile
Profile
Container x day aggregate
node: C
container: Z
service: S
Profile
Profile
Profile
Container x day aggregate
service: S
Service aggregate
Service aggregate
Service aggregate
Service x day aggregate

```
$ perforator fetch '{
    service = "yandex.search",
    timestamp > "10:00",
    node_id = "server123.yandex.net"
}' -o flamegraph.html
```

Perforator. Как использовать?

Cluster-wide профилировщик сделан. Как применять?

Perforator. Как использовать?

Cluster-wide профилировщик сделан. Как применять?

- Отладка проблем с производительностью в прошлом

Perforator. Как использовать?

Cluster-wide профилировщик сделан. Как применять?

- Отладка проблем с производительностью в прошлом
- CPU на строчку: сколько каждая строчка сжигает CPU?

Perforator. Как использовать?

Cluster-wide профилировщик сделан. Как применять?

- Отладка проблем с производительностью в прошлом
- CPU на строчку: сколько каждая строчка сжигает CPU?
- Wall-time profiling: где процесс тратит реальное время?

Perforator. Как использовать?

Cluster-wide профилировщик сделан. Как применять?

- Отладка проблем с производительностью в прошлом
- CPU на строчку: сколько каждая строчка сжигает CPU?
- Wall-time profiling: где процесс тратит реальное время?
- Чтение `thread_local` значений: профиль про A/B тестам

Perforator. Как использовать?

Cluster-wide профилировщик сделан. Как применять?

- Отладка проблем с производительностью в прошлом
- CPU на строчку: сколько каждая строчка сжигает CPU?
- Wall-time profiling: где процесс тратит реальное время?
- Чтение `thread_local` значений: профиль про A/B тестам
- Генерация профилей для AutoFDO. PGO без сложных стендов

Perforator. Как использовать?

Cluster-wide профилировщик сделан. Как применять?

- Отладка проблем с производительностью в прошлом
- CPU на строчку: сколько каждая строчка сжигает CPU?
- Wall-time profiling: где процесс тратит реальное время?
- Чтение `thread_local` значений: профиль про A/B тестам
- Генерация профилей для AutoFDO. PGO без сложных стендов
- Minicores: записываем легковесные стектрейсы при фатальных сигналах (включая SIGKILL!)

Что дальше?

Что дальше?

- Верим, что система будет полезна миру

Что дальше?

- Верим, что система будет полезна миру
- Планируем в 2024 году выйти в OpenSource

Что дальше?

- Верим, что система будет полезна миру
- Планируем в 2024 году выйти в OpenSource
- Ждите новостей!

Выводы

Выводы

- Между «вернуть frame pointers» и «отказаться от профилирования» можно не выбирать

Выводы

- Между «вернуть frame pointers» и «отказаться от профилирования» можно не выбирать
- eBPF – новая глава в истории анализа производительности

Выводы

- Между «вернуть frame pointers» и «отказаться от профилирования» можно не выбирать
- eBPF – новая глава в истории анализа производительности
- DWARF не катастрофически сложный

Выводы

- Между «вернуть frame pointers» и «отказаться от профилирования» можно не выбирать
- eBPF – новая глава в истории анализа производительности
- DWARF не катастрофически сложный
- Perforator едет в OpenSource!

Спасибо!

Будем рады ответить на ваши вопросы