Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем

Полынцов Михаил Александрович

Реализация алгоритма FastFDs в платформе Desbordante

Отчёт по учебной практике

Научный руководитель:
ассистент кафедры ИАС Чернышев Г. А.

Санкт-Петербург
2022

Оглавление

Введение

Функциональные зависимости (ФЗ) задают корреляцию между атрибутами фиксированного отношения (в терминах реляционной модели [6]). Поиск ФЗ, которые удерживаются над отношением — важная задача во многих областях анализа данных. ФЗ представляют собой метаданные (информацию о данных) и используются для интеллектуального анализа данных (data mining [3]), очистки данных (data cleansing [9]), интеграции данных (data integration [9]) и т. д. В связи с этим существует множество алгоритмов автоматического поиска ФЗ, подробный обзор которых представлен в работах [1, 8].

Эффективность алгоритма поиска ФЗ зависит [4] от входных данных. Какие-то алгоритмы наиболее эффективны, например, при работе с таблицами, где мало атрибутов, но большое количество кортежей, какие-то наоборот. Таким образом, возникла необходимость создания платформы, где собраны алгоритмы поиска ФЗ и предоставлен интерфейс для их использования. Такой платформой-первопроходцем стал Metanome [5]. Бэкенд Metanome и соответственно все алгоритмы реализованы на Java. Это влечет существенные проблемы, связанные с производительностью и пространством возможных оптимизаций (SIMD и т. д.), описанные в работе [7]. Чтобы решить эти проблемы был создан фреймворк высокопроизводительного поиска ФЗ — Desbordante [7], реализованный на языке C++.

На момент написания данной работы Desbordante активно развивается и нуждается в реализации основных алгоритмов поиска ФЗ. В данной работе представлен отчет о реализации алгоритма FastFDs [16] в платформе Desbordante.

1 Постановка задачи

Цель данной учебной практики — реализация алгоритма поиска функциональных зависимостей FastFDs в платформе Desbordante на языке программирования C++ и исследование его производительности. Для достижения этой цели были поставлены следующие задачи:

- Спроектировать и реализовать версию алгоритма FastFDs из оригинальной работы [16] в Desbordante;
- Реализовать оптимизации, использованные в Metanome;
- Исследовать возможности распараллеливания FastFDs и реализовать многопоточную версию;
- Сравнить производительность реализованного алгоритма с реализацией из Metanome.

2 Обзор предметной области

2.1 Терминология

Приведем необходимые определения из работ [4, 16].
Пусть R — заголовок отношения и r — тело отношения.

Определение 1. Пусть $X, Y \subseteq R$.

1. $X \to Y$ — функциональная зависимость над r если и только если $\forall t_1, t_2 \in r,\ t_1[X] = t_2[X] \Rightarrow t_1[Y] = t_2[Y]$.

2. $X \to Y$ тривиальная ФЗ над r если и только если $Y \subseteq X$.

3. $X \to Y$ минимальная ФЗ над r если и только если $\forall Z \subsetneq X :\ Z \to Y$ не ФЗ над r.

Определение 2.

1. Пусть $t_1, t_2 \in r$, *набор расхождений* (*difference set*) от t_1 и t_2, есть

$$D(t_1, t_2) := \{B \in R \mid t_1[B] \neq t_2[B]\}.$$

2. *Наборы расхождений* (*difference sets*) над r есть

$$\mathfrak{D}_r := \{D(t_1, t_2) \mid t_1, t_2 \in R,\ D(t_1, t_2) \neq \varnothing\}.$$

3. Пусть $A \in R$, *наборы расхождений по атрибуту* A (*difference sets modulo* A) есть

$$\mathfrak{D}_r^A := \{D \backslash \{A\} \mid D \in \mathfrak{D}_r,\ A \in D\}.$$

4. *Минимальные наборы расхождений по атрибуту* A есть

$$\underline{\mathfrak{D}_r^A} := \{D \in \mathfrak{D}_r^A \mid D' \in \mathfrak{D}_r^A,\ D' \subseteq D \Rightarrow D' = D\}.$$

Определение 3. Пусть 2^R есть булеан множества R и $\mathcal{X} \subseteq 2^R$. Тогда

1. $X \subseteq R$ есть покрытие (покрывает) $\mathcal{X}$ если и только если

$$\forall Y \in \mathcal{X},\ Y \cap X \neq \varnothing.$$

2. X есть минимальное покрытие $\mathcal{X}$ если и только если

$$\forall Z,\ Z \subsetneq X,\ Z \text{ не покрывает } \mathcal{X}.$$

Теорема 1.
Пусть $X \subseteq R,\ A \in R \backslash X$. $X \to A$ минимальная ФЗ над r тогда и только тогда, когда X есть минимальное покрытие $\mathfrak{D}_r^A$.

Замечание.
Любое покрытие $\underline{\mathfrak{D}_r^A}$ так же есть покрытие $\mathfrak{D}_r^A$. Следовательно, чтобы найти минимальные покрытия $\mathfrak{D}_r^A$, достаточно найти минимальные покрытия $\underline{\mathfrak{D}_r^A}$.

Определение 4.
1. Пусть $t_1,\ t_2 \in r$, *набор совпадений* (*agree set*) от t_1 и t_2, есть

$$A(t_1, t_2) := \{B \in R \mid t_1[B] = t_2[B]\}.$$

2. *Наборы совпадений* (*agree sets*) над r есть

$$A_r := \{A(t_1, t_2) \mid t_1, t_2 \in r\}.$$

Лемма 1.
Пусть $A \in R$, тогда $\mathfrak{D}_r = \{R \backslash X \mid X \in A_r\}$.

Определение 5. Пусть $A \in R$, тогда
1. $t_1, t_2 \in r$ эквивалентны по модулю A ($t_1 \sim_A t_2$) если и только если $t_1[A] = t_2[A]$.
2. Пусть $t \in r$, обозначим за $[t]_A$ класс эквивалентности кортежа t по отношению эквивалентности $\sim_A$. Тогда *партиция от A* (*stripped partition of A*) есть

$$\pi_A := \{[t]_A \mid t \in R\ \ |[t]_A| > 1\}.$$

3. Пусть Π_r есть множество все партиций над r, тогда *максимальное представление* (*maximal representation*) есть

$$\overline{\Pi_r} := \{\pi \in \Pi_r \mid \pi' \in \Pi_r, \pi \subseteq \pi' \Rightarrow \pi' = \pi\}.$$

Лемма 2.
Пусть $t_1, t_2 \in r$. Если не существует такого $\pi \in \overline{\Pi_r}$, что $t_1, t_2 \in \pi$, то $\forall A \in R: \ t_1[A] \neq t_2[A]$.

Следствие.
Таким образом, чтобы найти A_r, достаточно найти набор совпадений для всех пар кортежей из каждого класса эквивалентности в $\pi \in \overline{\Pi_r}$ для всех таких π.

2.2 Алгоритм FastFDs

Алгоритм FastFDs ищет минимальные нетривиальные ФЗ с правой частью, состоящей из одного атрибута. Все остальные ФЗ над тем же отношением могут быть найдены с помощью правил вывода Армстронга [14]. Используя теорему 1 алгоритм сводит проблему поиска ФЗ к проблеме поиска минимальных покрытий $\mathfrak{D}_r^A$ (то есть $\underline{\mathfrak{D}_r^A}$) для всех A из R. Алгоритм работает следующим образом:

1. Вычисляет Π_r и $\overline{\Pi_r}$.
2. Находит наборы совпадений A_r, используя результат леммы 2.
3. Находит наборы расхождений $\mathfrak{D}_r$, используя результат леммы 1.
4. Находит минимальные наборы расхождений по A $\underline{\mathfrak{D}_r^A}$ для всех A из R.
5. Находит все минимальные покрытия $\underline{\mathfrak{D}_r^A}$ для каждого A из R. Тем самым, находит все минимальные нетривиальные ФЗ с одним атрибутом в правой части, что и требовалось.

Для поиска минимальных покрытий $\underline{\mathfrak{D}_r^A}$ используется дерево поиска с упорядоченным множеством атрибутов в узлах. Каждый узел содержит столько детей, сколько атрибутов (каждому ребенку соответствует атрибут в родителе). Путь в дереве от корня до листа есть кандидат на минимальное покрытие. Дерево обходится поиском в глубину. Атрибуты в каждом узле упорядочены по убыванию таким образом: атрибут

A больше атрибута B, если A покрывает больше наборов расхождений, чем B.

Более подробное описание алгоритма можно найти в работе [16].

2.3 Реализация в Metanome

В Metanome не используется замечание к теореме 1 и покрытия ищут напрямую у наборов расхождений по атрибуту A D_r^A, минимальные наборы расхождений по атрибуту A $\underline{\mathfrak{D}_r^A}$ вообще не вычисляются. Что выглядит упущением, поскольку в работе [4] сказано, что в реализации алгоритмов авторы старались следовать оригинальному описанию как можно более точно.

Набор совпадений от двух кортежей в оригинальной работе по FastFDs предлагается находить просто проверяя, в каких атрибутах кортежи совпадают. В Metanome для вычисления набора совпадений от двух кортежей реализован более эффективный [11] подход, использующий структуру данных *идентифицирующий набор* (*identifier set*). Утверждается, что получать набор совпадений пересекая идентифицирующие наборы кортежей более эффективно, чем сравнивать кортежи поатрибутно.

Для вычисления наборов совпадений и максимального представления в Metanome существует несколько различных методов, все они были реализованы в Desbordante автором данной работы. Выбрать метод, который будет использоваться, можно на этапе компиляции..

2.4 Распараллеливание

В оригинальной работе [16] описывается только однопоточная версия FastFDs, а распараллеливание относят к направлениям дальнейшей работы. В ней, однако, упоминается очевидный вариант распараллеливания — выполнять вычисление $\mathfrak{D}_r^A$ и его покрытий для разных A параллельно (они полностью независимы). Проведенный на момент написания данной работы обзор показал, что работ посвященных многопоточной версии FastFDs не появилось. Существуют работы [12, 13],

посвященные распределенному поиску ФЗ. Они не представляет интереса, так как алгоритмы в них проектировались с идеей о том, что обмен данными может являться узким местом. В то время как потоки на одном компьютере работают в условиях архитектуры с полным разделением ресурсов (shared-everything), где обмен данными реже является узким местом, чем, например, в архитектуре без разделения ресурсов (shared-nothing).

В работе [4], посвященной реализации и сравнению производительности алгоритмов в Metanome, параллельность не упоминается. В исходном коде FastFDs в Metanome, тем не менее, присутствует многопоточный код. А именно, реализовано многопоточное вычисление:

- максимального представления $\overline{\Pi_r}$;
- наборов совпадений A_r из $\overline{\Pi_r}$;
- наборов расхождений $\mathfrak{D}_r$ из A_r.

В Metanome реализован многопоточный алгоритм вычисления $\overline{\Pi_r}$. В Metanome не распараллелен цикл, который вычисляет $\mathfrak{D}_r^A$ и его покрытия для всех $A \in R$. Это является недочетом реализации, при реализации алгоритма автором данной работы этот недочет был исправлен. Кроме всего выше описанного в алгоритме FastFDs нет мест, которые можно эффективно распараллелить.

В многопоточной реализации FastFDs в Metanome используется потокобезопасная структура данных `ConcurrentHashMap`[1], вместо `HashSet`[2] и `HashMap`[3], которые используются в однопоточной версии. Для каждого алгоритма из перечисленных выше создается пул потоков с помощью вызова метода `Executors.newFixedThreadPool(int)`[4]. Для реализации FastFDs в Desbordante была использована структура пула пото-

[1] docs.oracle.com/javase/8/docs/api/java/util/concurrent/ConcurrentHashMap.html (дата обращения: 02.12.2021)

[2] docs.oracle.com/javase/8/docs/api/java/util/HashSet.html (дата обращения: 02.12.2021)

[3] docs.oracle.com/javase/8/docs/api/java/util/HashMap.html (дата обращения: 02.12.2021)

[4] docs.oracle.com/javase/7/docs/api/java/util/concurrent/Executors.html#newFixedThreadPool(int) (дата обращения: 02.12.2021)

ков `thread_pool` из библиотеки `Boost.Asio`[5]. Была выбрана именно эта библиотека, чтобы избежать новых зависимостей, так как в Desbordante уже используются библиотеки из Boost. Также Boost известен тем, что библиотеки в нем качественно реализованы и долго поддерживаются, соответственно искать альтернативы не было необходимости. Стандартная библиотека C++ не предоставляет потокобезопасных структур данных. Существуют библиотеки, реализующие необходимые структуры: libcds[6], junction[7], xenium[8], ck[9]. Они, однако, не были использованы по нижеописанным причинам.

В тех частях алгоритма, где можно эффективно использовать параллельность, разные потоки выполняют независимый или почти независимый код. Единственная возникающая зависимость: потокам необходимо писать выходные данные в один и тот же контейнер (`std::vector` или `std::unordered_set`). Для того, чтобы избежать гонок, было реализовано следующее решение. Каждый поток работает с отдельным контейнером, а по завершении работы всех потоков, выполняется слияние возвращенных результатов в один контейнер. Также структуры данных, предоставляемые библиотеками, имеют несовместимый интерфейс с контейнерами стандартной библиотеки, что усложняет написание обобщенного кода. То есть, чтобы эффективно использовать потокобезопасные структуры данных вместе со стандартными, необходимо реализовать промежуточный класс, унифицирующий их интерфейсы. При этом не ясно, будет ли прирост в производительности при использовании потокобезопасных структур выше, чем при использовании подхода со слиянием.

Таким образом, было принято решение отказаться от потокобезопасных структур из сторонних библиотек. Исследование их влияния на производительность является направлением дальнейшей работы.

[5] boost.org/doc/libs/1_77_0/doc/html/boost_asio.html (дата обращения: 02.12.2021)
[6] github.com/khizmax/libcds (дата обращения: 03.12.2021)
[7] github.com/preshing/junction (обращения: 03.12.2021)
[8] github.com/mpoeter/xenium (дата обращения: 03.12.2021)
[9] github.com/concurrencykit/ck (дата обращения: 03.12.2021)

3 Реализация алгоритма

3.1 Общая архитектура

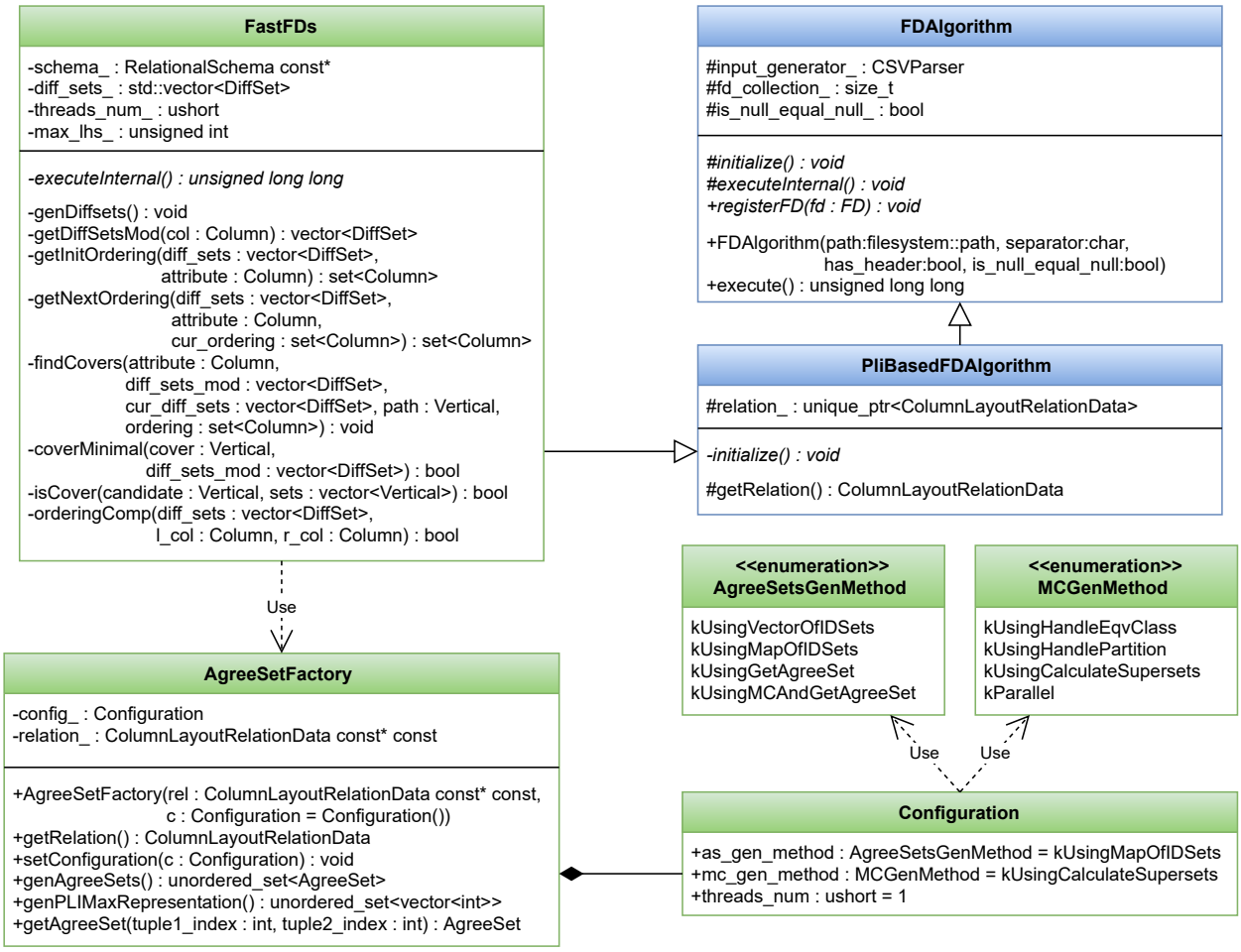

Рис. 1: Иерархия классов.

Иерархию классов алгоритма можно видеть на Рис. 1, опущены некоторые детали, зеленым отмечены классы, реализованные автором данной работы. Все основные методы алгоритма повторяют псевдокод, представленный в работе [16], поэтому их подробное описание будет опущено.

Все алгоритмы начинают свое выполнение с парсинга входной таблицы. Класс `CSVParser` представляет собой парсер и уже реализован в Desbordante. Парсинг таблицы, хранение найденных ФЗ и некоторая другая общая вспомогательная функциональность вынесена в класс `FDAlgorithm`. FastFDs и многие другие алгоритмы, например, Tane [15],

Pyro [10], FD_Mine [17], DFD [2] работают с таблицей через множество всех партиций Π_r. Вычисление Π_r вынесено в класс `PliBasedFDAlgorithm` и все перечисленные алгоритмы наследуются от него.

Класс `FDAlgorithm` имеет абстрактный метод `executeInternal()`, который вызывается внутри `execute()` после инициализации через `initialize()`. Это единственный метод, который необходимо было реализовать в классе `FastFDs`.

Вычисление наборов совпадений A_r вынесено в отдельный класс `AgreeSetFactory`, так как A_r является промежуточным шагом алгоритма Dep-Miner [11]. Вынесение этого шага в отдельный класс помогает избежать дублирования кода. Также это помогает отделить логику непосредственного вычисления ФЗ от логики вычисления вспомогательных структур. Объекту класса `AgreeSetFactory` можно задать конфигурацию, в которой указать количество используемых потоков, алгоритм вычисления максимального представления $\overline{\Pi_r}$ и алгоритм вычисления самих наборов совпадений. Все алгоритмы вычисления $\overline{\Pi_r}$ взяты из исходного кода Metanome и задаются перечислением `MCGenMethod`. Перечисление `AgreeSetsGenMethod` задает алгоритмы вычисления наборов совпадений:

- `kUsingMapOfIDSets`, алгоритм описанный в работе [11] и реализованный (уточненный) в Metanome;
- `kUsingVectorOfIDSets`, алгоритм пересекающий все пары индентифицирующих наборов напрямую, а не через пары элементов максимального представления, как в `kUsingMapOfIDSets`;
- `kUsingGetAgreeSet`, алгоритм из оригинальной работы [16];
- `kUsingMCAndGetAgreeSet`, алгоритм из оригинальной работы [16], однако итерируется не по всем партициям целиком, а только по максимальному представлению.

3.2 Многопоточная версия

Распараллеливание FastFDs сводится к распараллеливанию основных циклов в логических частях алгоритма. Сами части (генерация партиций, генерация наборов совпадений и т. д.) должны быть последовательными относительно друг друга, потому что каждая работает с результатами предыдущей.

По существу были распараллелены все основные циклы шагов алгоритма, описанные в 2.2:

- внешний цикл по максимальному представлению во время генерации наборов совпадений;
- цикл по сгенерированным наборам совпадений при вычислении наборов расхождений;
- цикл по атрибутам, вычисляющий множество минимальных наборов расхождений по атрибуту $\underline{\mathfrak{D}_r^A}$ и его минимальные покрытия.

Генерация партиций не была затронута, поскольку была реализована не автором данной работы. Шаги вычисления $\underline{\mathfrak{D}_r^A}$ и его минимальных покрытий выполняются последовательно, так как использование нескольких потоков здесь привело бы к слишком большому общему количеству потоков и потере производительности.

4 Эксперименты

4.1 Описания экспериментов

Эксперименты проводились следующим образом: было сделано десять запусков алгоритма в Metanome и Desbordante на каждом наборе данных, взято среднее значение с доверительные интервалом 95%, результаты отображены на графике. Desbordante компилировался с флагом оптимизаций `-O3`. Обе реализации Metanome и Desbordante замеряют время работы алгоритма (без учета парсинга входного файла) и выводят на экран, это время и использовалось как показатель производительности. Было проведено два отдельных эксперимента: сравнение однопоточных и многопоточных версий. Многопоточным версиям указывалось 4 потока, что равно количеству физических ядер умноженному на количество гиперпотоков на каждом ядре на машине, где проводились эксперименты. Чтобы минимизировать влияние операционной системы на результаты, эксперименты запускались на незагруженной системе (без других тяжеловесных процессов), между итерациями очищались кеши путем записи "3" в `/proc/sys/vm/drop_caches`, файл подкачки отключался командой `swapoff -a`. При запуске Metanome с несколькими потоками виртуальной машине Java выделялось пять гигабайт под максимальный размер кучи (больше физически нельзя было выделить в оперативной памяти).

Характеристики системы, на которой проводились эксперименты: Intel Core i5-7200U CPU @ 2.50GHz × 4, 8 GiB RAM, 240GB KINGSTON SA400S3, Ubuntu 20.04.2 LTS, Kernel 5.11.0-41-generic. Metanome запускался на OpenJDK версии 11.0.11 (LTS) с настройками по умолчанию (кроме размера кучи).

4.2 Угрозы валидности

На производительность реализации из Metanome могут влиять:

- тип виртуальной машины Java (GraalVM позволяет использовать

Ahead Of Time компиляцию);

- версия JDK;
- тип сборщика мусора (Garbage-First, MarkSweep и т. д.).

В работе [7] утверждается, что лучшую производительность Metanome можно получить, используя LTS версию JDK и оставить настройки виртуальной машины по умолчанию, что и было сделано при проведении экспериментов.

4.3 Результаты

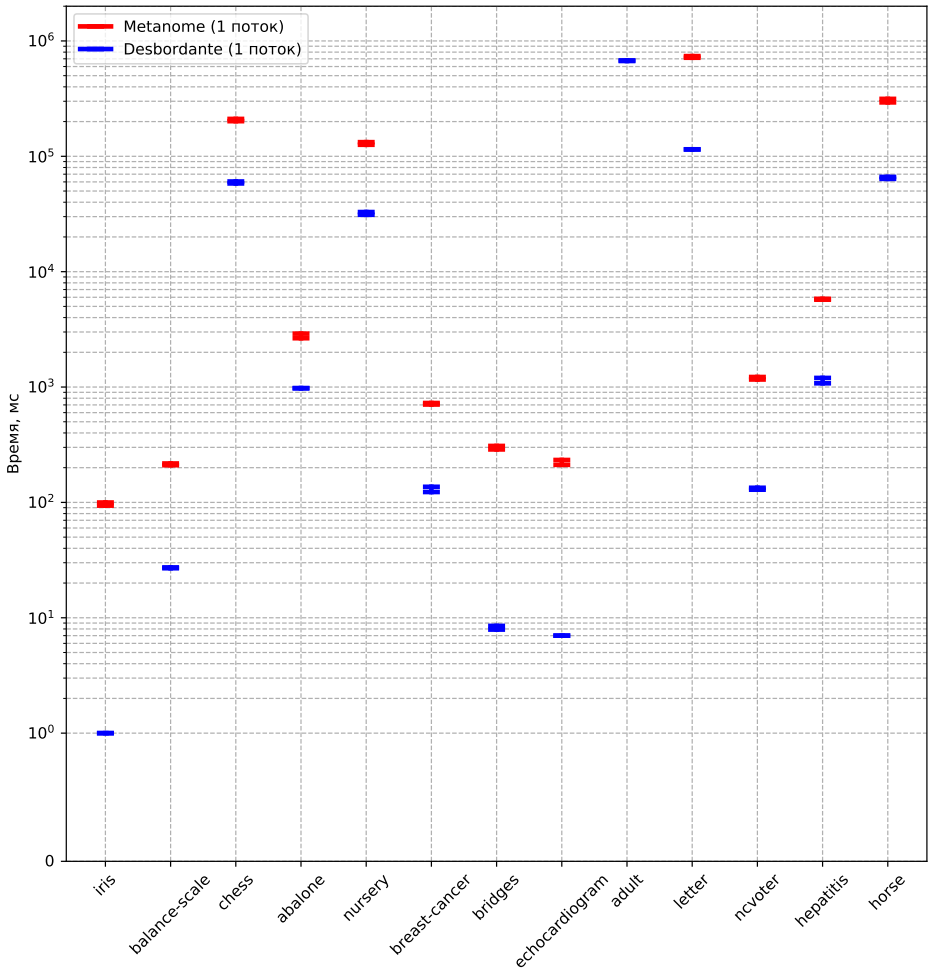

Рис. 2: Результаты (1 поток).

Результаты экспериментов можно видеть на Рис. 2 и Рис. 3. Однопоточная версия Metanome превысила предел по времени в один час на наборе данных `adult`, поэтому результат на этом наборе не отображен на Рис. 2. Многопоточная версия Metanome превысила предел по памяти в семь гигабайт на наборах данных `chess, nursery, adult, letter`, поэтому результат на этих наборах не отображен на Рис. 3.

Можно видеть, что реализация в Desbordante на порядок быстрее. Также видно, что многопоточная версия реализации в Metanome практически не выигрывает в производительности однопоточную, в то время как многопоточная версия в Desbordante в среднем в полтора раза быстрее однопоточной.

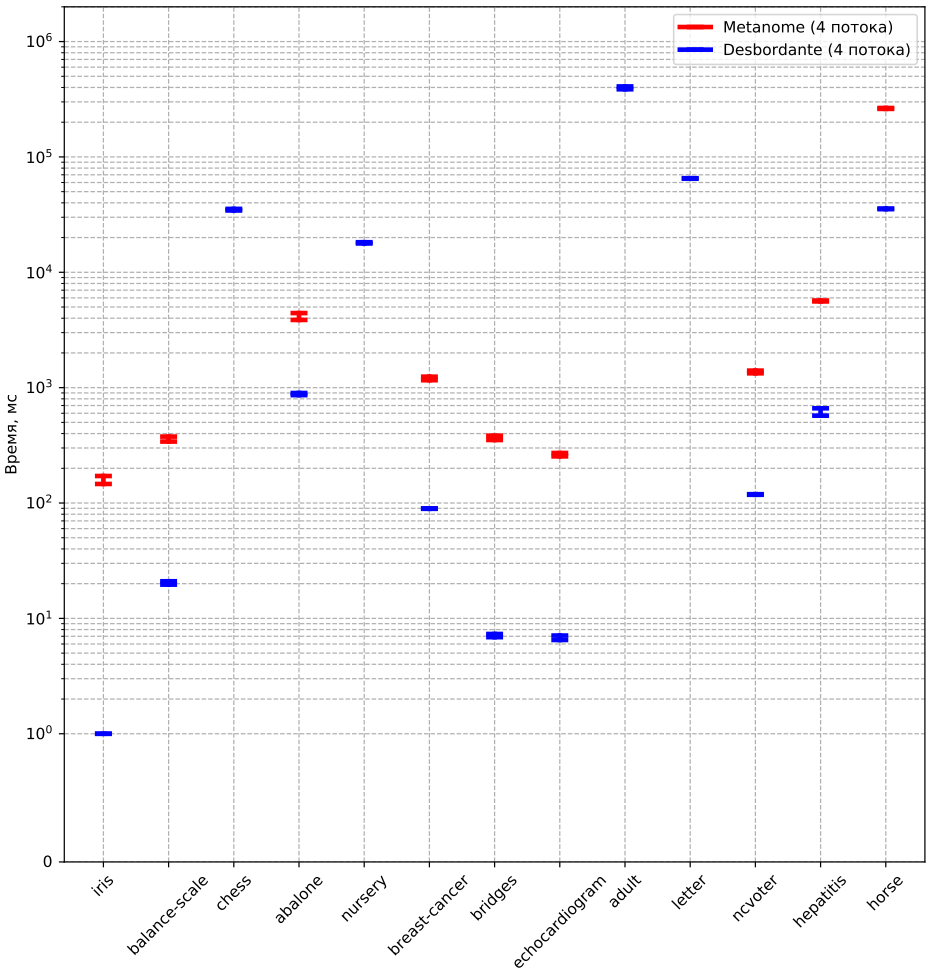

Рис. 3: Результаты (4 потока).

5 Заключение

В рамках данной работы был реализован алгоритм поиска функциональных зависимостей FastFDs в платформе Desbordante и исследована его производительность. Для достижения этой цели были решены следующие задачи:

- Спроектирована и реализована версия алгоритма FastFDs из оригинальной работы [16] в Desbordante (исходный код доступен на Github[10], имя пользователя polyntsov);
- Реализованы оптимизации, использованные в Metanome;
- Исследованы возможности распараллеливания FastFDs и реализована многопоточная версия;
- Проведено сравнение производительности реализованного алгоритма с реализацией FastFDs из Metanome.

[10] github.com/Mstrutov/Desbordante (дата обращения: 15.12.2021)

Список литературы

[1] Abedjan Ziawasch, Golab Lukasz, and Naumann Felix. Profiling Relational Data: A Survey // The VLDB Journal. — 2015. — aug. — Vol. 24, no. 4. — P. 557–581. — Access mode: https://doi.org/10.1007/s00778-015-0389-y.

[2] Abedjan Ziawasch, Schulze Patrick, and Naumann Felix. DFD: Efficient Functional Dependency Discovery // Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management. — New York, NY, USA : Association for Computing Machinery. — 2014. — CIKM '14. — P. 949–958. — Access mode: https://doi.org/10.1145/2661829.2661884.

[3] Abedjan Ziawasch, Golab Lukasz, Naumann Felix, and Papenbrock Thorsten. Data Profiling. — First ed. — Morgan & Claypool Publishers, 2018. — Nov. — Vol. 10 of Synthesis Lectures on Data Management.

[4] Papenbrock Thorsten, Bergmann Tanja, Finke Moritz, Zwiener Jakob, and Naumann Felix. Data Profiling with Metanome // Proc. VLDB Endow. — 2015. — aug. — Vol. 8, no. 12. — P. 1860–1863. — Access mode: https://doi.org/10.14778/2824032.2824086.

[5] Papenbrock Thorsten, Bergmann Tanja, Finke Moritz, Zwiener Jakob, and Naumann Felix. Data Profiling with Metanome // Proc. VLDB Endow. — 2015. — aug. — Vol. 8, no. 12. — P. 1860–1863. — Access mode: https://doi.org/10.14778/2824032.2824086.

[6] Date C.J. An Introduction to Database Systems. — 8 ed. — USA : Addison-Wesley Longman Publishing Co., Inc., 2003. — ISBN: 0321197844.

[7] Strutovskiy Maxim, Bobrov Nikita, Smirnov Kirill, and Chernishev George A. Desbordante: a Framework for Exploring Limits of Dependency Discovery Algorithms // 29th Conference of

Open Innovations Association, FRUCT 2021, Tampere, Finland, May 12-14, 2021. — IEEE. — 2021. — P. 344–354. — Access mode: https://doi.org/10.23919/FRUCT52173.2021.9435469.

[8] Liu Jixue, Li Jiuyong, Liu Chengfei, and Chen Yongfeng. Discover Dependencies from Data—A Review // IEEE Transactions on Knowledge and Data Engineering. — 2012. — Vol. 24, no. 2. — P. 251–264.

[9] Ilyas Ihab F. and Chu Xu. Data Cleaning. — New York, NY, USA : Association for Computing Machinery, 2019. — ISBN: 978-1-4503-7152-0.

[10] Kruse Sebastian and Naumann Felix. Efficient Discovery of Approximate Dependencies // Proc. VLDB Endow. — 2018. — mar. — Vol. 11, no. 7. — P. 759–772. — Access mode: https://doi.org/10.14778/3192965.3192968.

[11] Lopes Stéphane, Petit Jean-Marc, and Lakhal Lotfi. Efficient Discovery of Functional Dependencies and Armstrong Relations. — 2000. — 03. — Vol. 1777. — P. 350–364.

[12] Saxena Hemant, Golab Lukasz, and Ilyas Ihab. Distributed Discovery of Functional Dependencies. — 2019. — 04. — P. 1590–1593.

[13] Saxena Hemant, Golab Lukasz, and Ilyas Ihab F. Distributed Implementations of Dependency Discovery Algorithms // Proc. VLDB Endow. — 2019. — jul. — Vol. 12, no. 11. — P. 1624–1636. — Access mode: https://doi.org/10.14778/3342263.3342638.

[14] Silberschatz Abraham, Korth Henry, and Sudarshan S. Database Systems Concepts. — 5 ed. — USA : McGraw-Hill, Inc., 2005. — ISBN: 0072958863.

[15] Huhtala Ykä, Kärkkäinen Juha, Porkka Pasi and Toivonen Hannu. Tane: An Efficient Algorithm for Discovering Functional and

Approximate Dependencies // Computer Journal. — 1999. — Vol. 42, no. 2. — P. 100–111.

[16] Wyss Catharine, Giannella Chris, and Robertson Edward. FastFDs: A Heuristic-Driven, Depth-First Algorithm for Mining Functional Dependencies from Relation Instances Extended Abstract. — 2001. — 01. — P. 101–110.

[17] Yao Hong, Hamilton Howard, and Butz Cory. FD_Mine: Discovering Functional Dependencies in a Database Using Equivalences. — 2002. — 01. — P. 729–732.