Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский Государственный Университет»

Математическое обеспечение и администрирование информационных
систем

Севостьянов Всеволод Олегович

Исследование и разработка алгоритмов фрагментирования для RDF графов

Магистерская диссертация

Научный руководитель:
ассистент каф. ИАС Чернышев Г.А.

Рецензент:
Смирнов К.К.

Санкт-Петербург
2017

SAINT-PETERSBURG STATE UNIVERSITY

Software and Administration of Information Systems

Vsevolod Sevostyanov

Research and development of RDF graph partitioning algorithms

Master's Thesis

Scientific supervisor:
assistant professor George Chernishev

Reviewer:
Kirill Smirnov

Saint-Petersburg
2017

Оглавление

1. Введение

Semantic Web представляет собой популярную область исследований, не последнее место в которой занимают задачи работы с данными. Консорциумом W3C[1] был разработан RDF [9] — фреймворк, который предлагает формат представления данных, основанный на тройках. Каждая тройка имеет вид (S, P, O), где S обозначает субъект, O — объект, а P — отношение (предикат) между ними. Такой способ представления позволяет также воспринимать набор данных как граф, где вершины — объекты и субъекты, а дуги — отношения. Пример тройки представлен в Таблице 1, это же утверждение в формате RDF/XML представлено в Листинге 1.

Запросы к данным RDF осуществляются на языке SPARQL [14], который позволять определять интересующие пользователя данные с помощью шаблонов. Тройка будет включена в ответ в том случае, если она соответствует предоставленному шаблону. Структура запросов такова, что каждый из них можно представить в виде шаблонного графа, и выполнение запроса будет заключаться в сопоставлении графа RDF шаблону запроса. Пример простого запроса приведен в Листинге 2, пример простого шаблона — на Рисунке 2.

```
<?xml version=''1.0'' encoding=''utf-8'' ?>
<rdf:RDF xmlns:rdf=''http://www.w3.org/1999/02/22-rdf-syntax-ns#''
xmlns:foaf=''http://xmlns.com/foaf/0.1/''>

<rdf:Description rdf:nodeID=''a''>
  <foaf:name>Johnny Lee Outlaw</foaf:name>
  <foaf:mbox rdf:resource=''mailto:jlow@example.com''/>
</rdf:Description>

<rdf:Description rdf:nodeID=''b''>
  <foaf:name>Peter Goodguy</foaf:name>
  <foaf:mbox rdf:resource=''mailto:peter@example.org''/>
</rdf:Description>
</rdf:RDF>
```

Листинг 1: Пример документа на RDF/XML

[1] https://www.w3.org

Субъект	Предикат	Объект
a	Name	Johnny Lee Outlaw
a	Mail Box	jlow@example.com
b	Name	Peter Goodguy
b	Mail Box	peter@example.org

Таблица 1: Простое утверждение

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE
{ ?x foaf:name ?name .
  ?x foaf:mbox ?mbox }
```

Листинг 2: Простой SPARQL запрос

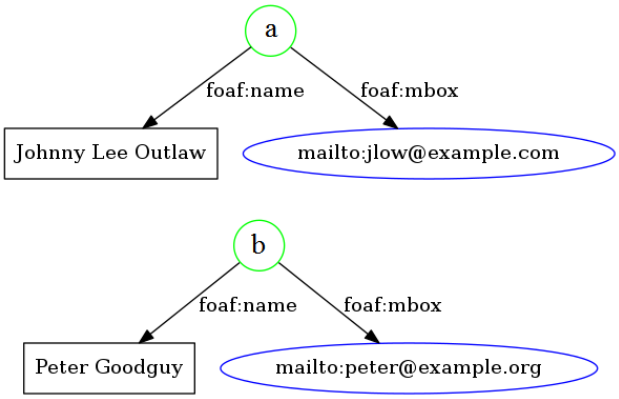

Рис. 1: Пример графового представления RDF

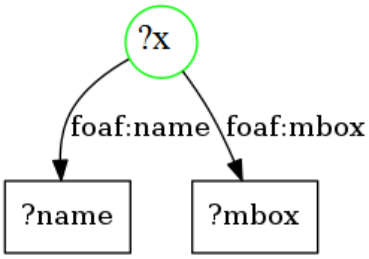

Рис. 2: Пример простого шаблона

2. Постановка задачи

2.1. Мотивация работы

Количество троек в наборе данных (дуг в графе) может достигать нескольких миллионов, поэтому встает вопрос об их хранении и эффективной обработке запросов к таким объемам информации. Многие работы в данной области работают именно с графами, тогда как некоторые предлагают способы трансляции данных RDF в реляционные таблицы, которые в общем случае более эффективны, чем троичные и графовые системы.

При рассмотрении работ складывается впечатление, что многие подходы к физической организации данных RDF не учитывают неравномерность распределения запросов по отношению к данным, в свете чего интересно рассмотреть стратегии фрагментирования, которые могли бы устранить или сгладить такую неравномерность.

2.2. Постановка задачи

Среди существующих подходов, которые будут рассмотрены в следующей секции, можно выделить [4], использующий так называемые характеристические множества [12]. Авторы предлагают способ трансляции RDF графа в реляционные таблицы, что позволяет быстро и эффективно выполнять запросы к данным, используя средства реляционных БД. Однако в работе не упоминается нагрузка на полученные таблицы, и не приводится сведений о качестве фрагментирования: не получаются ли фрагменты слишком крупными, не появляется ли неравномерность в распределении обращений и т.д.

Целью данной работы является проверка предположения, что достигнутое с помощью алгоритма [4] разделение на таблицы не всегда является лучшим, и возможно более равномерно распределить нагрузку на узлы распределенной системы.

Для достижения данной цели были поставлены следующие задачи:

- осуществить обзор предметной области;
- выбрать из существующих реализацию подхода;
- инструментировать выбранную реализацию функциями сбора информации;
- провести испытания на стандартном наборе данных;
- проанализировать полученную статистику и выяснить распределение запросов по отношению к фрагментам;
- оценить потенциальное влияние фрагментирования.

3. Обзор существующих подходов

При обращении с данными RDF как с графом, задача фрагментирования сводится к задачам сопоставления шаблонов и разреза графа. Среди существующих стратегий выбора разрезов стоит отметить [15, 16, 17].

В работе [15] основой для фрагментирования служат так называемые "шаблоны частого доступа" (frequent access patterns). Авторы анализируют запросы к базе данных, выделяя в них часто встречающиеся подграфы. Не все полученные подграфы послужат для выделения фрагментов: далее происходит выбор подмножества, удовлетворяющего ограничениям по хранению. В статье приведено доказательство $NP-$полноты данной проблемы, а также предлагаются жадные алгоритмы вертикального и горизонтального фрагментирования. Вертикальные фрагменты состоят из вершин и дуг, соответствующих конкретному "шаблону". При горизонтальном фрагментировании совпадения с шаблоном распределяются по разным фрагментам согласно введенному в работе минтерм-предикату.

Авторы [16] предлагают схожий подход для распределенных систем, учитывающий объемы взаимодействия между узлами кластера. Сначала для каждого запроса определяется мера его сегментированности относительно данного разделения на фрагменты как количество фрагментов, к которым придется обратиться для ответа на запрос. Далее решается задача поиска разделения, при котором мера сегментированности запросов, умноженная на частоту запроса, будет минимальна.

В статье [17] описывается система Trinity.RDF, в которой фрагментирование осуществляется на базе хеш-функции. Кортежи распределяются по фрагментам случайно, что позволяет в общем случае достичь приемлемых результатов.

Система METIS-2hops, представленная в статье [8], также использует фрагментирование графов. Данные разделяются на набор непересекающихся фрагментов, каждый из которых представляет собой множество близких друг к другу вершин. Затем, на основе данного разделе-

ния, происходит непосредственно распределение троек по физическим фрагментам с помощью “направленной n-шаговой гарантии” (directed n-hop guarantee): для набора вершин, определенных в какой-либо раздел, направленная 1-шаговая гарантия вычисляет все вершины в полном RDF графе, которые находятся на расстоянии одного шага от данного раздела и добавляет их в него. Направленная 2-шаговая гарантия берет в качестве раздела множества, полученные при вычислении 1-шаговой гарантии, и проделывает аналогичный шаг по присоединению смежных вершин к разделу. Продолжая аналогично, можно получить направленные гарантии для произвольного числа шагов. Как видно из названия системы, в ней используются 2-шаговые гарантии.

4. Характеристические множества

Авторы [4] предлагают алгоритм генерации реляционной схемы данных. Для формирования схемы используются характеристические множества [12]. Характеристическое множество сущности s — это множество дуг, выходящих из s. Нам сложно охарактеризовать сущность только по ее идентификатору, однако отношения, в которых состоит сущность, позволяют это сделать (см. Рис. 3[2]). Похожие характеристические множества чаще всего принадлежат семантически схожим сущностям, так что, сливая похожие характеристические множества, можно получать довольно подробные описания неких общих классов, которым принадлежат соответствующие сущности. Те сущности, которые состоят в малом количестве отношений, можно оставить в формате $(S,\ P,\ O)$ как таблицы иррегулярностей, их будет относительно немного по сравнению с количеством элементов в характеристических множествах. Между парой характеристических множеств может существвовать отношение, которое получается из наиболее частого предиката, включающего сущности из этих двух множеств. Полученные множества и отношения между ними представляют собой реляционные таблицы.

Алгоритм, согласно которому выделяются характеристические множества, заключается в следующем:

1. Для набора данных строятся характеристические множества и вычисляются их частоты, осуществляется поиск связей между множествами;

2. каждому характеристическому множеству сопоставляется метка, т.е. легко воспринимаемое человеком имя;

3. характеристические множества, которые структурно или семантически похожи, сливаются;

4. шаги 2 и 3 повторяются итеративно, пока не будет достигнута

[2]Рисунок взят из работы [12]

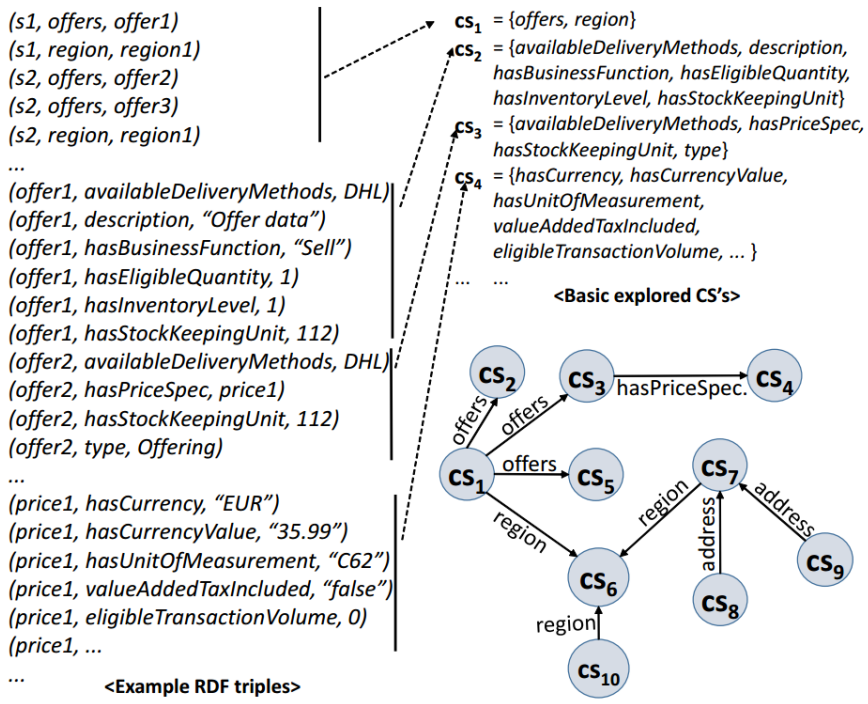

Рис. 3: Пример характеристических множеств и отношений между ними

граница схожести τ_{sim};

5. редко встречающиеся свойства удаляются из характеристических множеств;

6. отфильтровываются некоторые строки, чтобы поддержать однородность литералов и исключить ошибочные множественные значения атрибутов.

В результате получается набор характеристических множеств, представляющих собой реляционные таблицы, и таблиц "аномалий", содержащих тройки, которые не вошли или были исключены из характеристических множеств. Таблицы уже используются в качестве фрагментов. При выполнении запросов поиск осуществляется сначала в характеристических множествах, а затем в таблицах "аномалий".

5. Реализация в MonetDB

5.1. Архитектура MonetDB

MonetDB[3] — колоночная СУБД с открытым исходным кодом, разрабатываемая группой Мартина Керстена. Данные предтавляются в виде бинарных (состоящих из двух колонок) таблиц, BAT — Binary Association Table. Каждой такой таблице сопоставлен логический дескриптор BUN — Binary Unit.

В основе системы лежит бинарная алгебра, поддерживающая операторы select, join, semijoin, outerjoin, union, intersection, diff (для BAT и для отдельных колонок), агрегатные функции. Промежуточные результаты в системе всегда полностью материализуются в виде BAT.

Запросы к базе данных осуществляются на языке SQL 2003. Между SQL и ядром системы посредником выступает язык MAL — Monet Assembly Language.

В Листинге 3 приведен код, который будет сгенерирован при обработке запроса «select * from tmp;», где tmp — таблица, состоящая из одной колонки типа int и содержащая единственное значение 1000.

[3] `https://www.monetdb.org/`

```
function user.s9_1():void;
    X_32:void := querylog.define(''explain select * from tmp;'',''
   default_pipe'',23);
barrier X_42 := language.dataflow();
    X_16 := bat.new(nil:oid,nil:str);
    X_24 := bat.append(X_16,''sys.tmp'');
    X_19 := bat.new(nil:oid,nil:str);
    X_26 := bat.append(X_19,''number'');
    X_20 := bat.new(nil:oid,nil:str);
    X_27 := bat.append(X_20,''int'');
    X_21 := bat.new(nil:oid,nil:int);
    X_29 := bat.append(X_21,32);
    X_23 := bat.new(nil:oid,nil:int);
    X_31 := bat.append(X_23,0);
    X_1 := sql.mvc();
    C_2:bat[:oid] := sql.tid(X_1,''sys'',''tmp'');
    X_5:bat[:int] := sql.bind(X_1,''sys'',''tmp'',''number'',0);
    (C_8,r1_8) := sql.bind(X_1,''sys'',''tmp'',''number'',2);
    X_11:bat[:int] := sql.bind(X_1,''sys'',''tmp'',''number'',1);
    X_13 := sql.delta(X_5,C_8,r1_8,X_11);
    X_14 := algebra.projection(C_2,X_13);
exit X_42;
    sql.resultSet(X_24,X_26,X_27,X_29,X_31,X_14);
end user.s9_1;
```

Листинг 3: Пример кода на MAL

5.2. Исходный алгоритм

Выделение реляционной схемы в MonetDB/RDF реализован с помощью хранимых процедур, которые ссылаются на функции на языке C. Для разбора файла с данными необходимо вызвать процедуру rdf_shred, которая принимает на вход путь к файлу в формате .ttl или .rdf и создает в базе данных две таблицы: словарь $map0$, где каждому литералу сопоставлен идентификатор, и таблицу $spo0$, которая хранит тройки из идентификаторов. После этого необходимо загрузить метаданные онтологий с помощью скрипта $loadOntologyMetadata$, которые будут использованы для выведения типов сущностей. Чтобы непосредственно выделить реляционную схему, используется процеду-

ра $rdf\_reorganize$, которая следует описанному в предыдущей секции алгоритму. В системе MonetDB/RDF нет реализации языка SPARQL и для исполнения запросов нужно перевести их на язык SQL. Это делается с помощью базового функционала системы Virtuoso [5]. Перед началом выполнения запросов требуется создать дамп характеристических множеств, а затем вызвать процедуру $rdf\_prepare$, которая подготовит систему к выполнению запросов. Данная процедура назначает таблицам идентификаторы, которые совпадают с используемыми в Virtuoso. Для выполнения запросов с использованием реляционной схемы, каждый запрос необходимо предварить ключевым словом «sparql», иначе будет использовано троичное хранилище $spo0$.

5.3. Сбор статистики

Каждому характеристическому множеству в MonetDB/RDF соответствует уникальный идентификатор, который можно использовать для учета количества обращений к ним. В систему был встроен ассоциативный массив, который инициализируется сразу после выделения реляционной схемы. Каждый запрос, использующий реляционную схему, обращается к функции, которая возвращает идентификатор характеристического множества по его имени из запроса. При каждом обращении выполняется инкремент соответствующей переменной в ассоциативном массиве. В ходе выполнения запросов накапливается статистика, которую можно выгрузить в файл или сбросить с помощью специальных встроенных процедур.

6. Горизонтальное фрагментирование

Горизонтальное фрагментирование делит отношение на набор фрагментов, где каждый фрагмент хранит часть записей отношения. Благодаря этому уменьшается объем данных, которые необходимо обработать для выполнения запроса и повышается производительность СУБД. Чтобы горизонтальное фрагментирование считалось корректным [13], необходимо обеспечить восстановимость отношения, полноту и попарное непересечение фрагментов. К наиболее распространенным видам горизонтального фрагментирования относятся [18, 7]: (i) интервальное фрагментирование; (ii) фрагментирование с помощью хеш-функции; (iii) списковое фрагментирование [1] и (iv) колоночное фрагментирование [1].

При интервальном фрагментировании данные делятся на набор непересекающихся интервалов. Запись принадлежит какому-то фрагменту, если значение атрибута фрагментирования принадлежит соответствующему интервалу.

Фрагментирование с помощью хеш-функции заключается в применении этой функции к атрибутам фрагментирования. Запись относится к фрагменту на основе значения хеш-функции.

Для спискового фрагментирования необходимо каждому фрагменту сопоставить список допустимых значений атрибутов фрагментирования.

В случае колоночного фрагментирования в отношение добавляется виртуальный атрибут, значение которого задается формулой от некоторого набора атрибутов отношения. Далее для полученной колонки применяется некий метод фрагментирования.

Для горизонтального фрагментирования наиболее часто встречающихся характеристических множеств предлагается использовать интервальное разбиение по атрибуту наиболее частого предиката.

6.1. Горизонтальное фрагментирование в современных СУБД

В большинстве современных СУБД так или иначе используются различные методы фрагментирования.

Последняя на данный момент версия PostgreSQL поддерживает два: интервальное и списковое фрагментирование[4]. Отмечается, что выполнять разделение таблицы имеет смысл лишь тогда, когда иначе она будет иметь слишком большой размер. Точная граница размера таблицы, при котором фрагментирование принесет ощутимую пользу, определяется приложением, хотя как правило предельный размер таблицы равен количеству доступной физической памяти.

Колоночная СУБД MonetDB не имеет автоматического горизонтального фрагментирования данных, однако предоставляет возможность для их реализации с помощью MERGE TABLE[5], которая позволяет представить таблицу как объединение таблиц-фрагментов. Формирование фрагментов осуществляется вручную.

Система IBM DB2 реализует интервальное фрагментирование, основанное на хеш-функции и распределение по измерениям. Последний подход подразумевает совместное хранение похожих по значению в нескольких измерениях строк. Причем отмечается, что фрагментирование (ii) дает лучшую масштабируемость, (i) позволяет оперировать фрагментами целиком, а распределение по измерениям показывает наилучшую производительность выполнения запросов [6].

Oracle Database способна фрагментировать данные согласно интервалу, хеш-функции, списочно и составным методом. Среди преимуществ составного фрагментирования указывается потенциально большая эффективность прореживания, а также более мелкая гранулярность размещения данных [7]. СУБД предлагает также Partition Advisor: часть системы, которая способна предложить стратегию фрагментиро-

[4] https://www.postgresql.org/docs/9.6/static/ddl-partitioning.html

[5] https://www.monetdb.org/Documentation/Cookbooks/SQLrecipes/DataPartitioning

[6] https://www.ibm.com/developerworks/data/library/techarticle/dm-0608mcinerney

[7] http://docs.oracle.com/database/122/VLDBG/partition-concepts.htm

вания основываясь на наборе запросов к базе данных

6.2. Горизонтальное фрагментирование в исследовательских работах

Горизонтальное фрагментирование применяется в исследовательских работах различных областей, а также само остается предметом исследований.

Работа [11] предлагает SPOVC: масштабируемое хранилище RDF данных на основе PostgreSQL. Данные хранятся в колонках, соответствующих субъекту, предикату и объекту. Каждая колонка горизонтально фрагментирована согласно принципам PostgreSQL. Система показывает результаты, сравнимые с конкурентами в данной области.

Горизонтальное фрагментирование известно также в объектных СУБД как горизонтальное фрагментирование классов.

7. Испытания

7.1. Наборы данных

Испытания производились на синтетических данных Leigh Univesity Benchmark (LUBM) [6]. Данный эталонный тест производительности (benchmark) содержит информацию об университете: кафедры, студенты, преподаватели, исследовательские группы, публикации, курсы — и отношения между ними.

Эталонный тест производительности состоит из генератора набора сущностей, онтологии на языке OWL, набора запросов к данным и модуля тестирования.

Для измерений в данной работе использован пример сгенерированных данных, представленный на сайте проекта RDFox[8].

7.2. Запросы

Поскольку MonetDB/RDF не имеет собственной реализации языка SPARQL, возникла необходимость трансляции запросов на язык SQL. Для этих целей была использована СУБД Virtuoso [5]. В виду ограничений реализации MonetDB/RDF были исключены запросы DESCRIBE и CONSTRUCT, испытания производились исключительно на SELECT.

Для проведения измерений были использованы запросы для набора данных LUBM[9]. Были выбраны запросы 2, 4, 8 и 10. Первые три имеют явную звездчатую структуру. Запрос 10 основан на иерархии из онтологии, в нем использован общий класс «Student».

В большинстве запросов встречается очень мало констант и довольно большое количество переменных, такие же выводы можно сделать и по другим популярным наборам данных и запросам к ним[10].

[8] http://www.cs.ox.ac.uk/isg/tools/RDFox/2014/AAAI/

[9] http://swat.cse.lehigh.edu/projects/lubm/queries-sparql.txt

[10] https://km.aifb.kit.edu/projects/spartiqulator/examples.htm

7.3. Результаты

При анализе нагрузки выяснилось, что запросы наиболее часто обращаются к характеристическим множествам Graduate Student и Undergraduate Student, что отражено на Рисунке 5. Они относятся в том числе к классам Person и Student, запросы к которым составляют почти 60% от общего числа.

Для каждого характеристического множества системой подсчитывается его размер. Как видно из Рисунка 4, наиболее объемными оказались множества Publication, Undergraduate Student, Graduate Student и Teaching Course.

На Рисунке 6 показано соотношение между размером характеристического множества и количеством обращений к нему. Как можно видеть, наиболее часто используются одни из самых объемных таблиц: Undergraduate Student и Graduate Student. Это позволяет сделать вывод, что горизонтальное фрагментирование данных таблиц способно улучшить производительность системы путем предоставления возможности отсечения некоторых записей в случае фрагментирования на основе значений, а также просматривается возможность выравнивания нагрузки на данные в случае распределенной системы.

Для каждого запроса система выбирает таблицы, в которых потенциально содержатся записи, удовлетворяющие запросу. В Таблице 2 приведены характеристические множества, которые соответствуют запросам 2, 4, 8 и 10.

Исходя из размеров множеств, можно оценить объем информации для обработки. Разрешение Запроса 2 требует выполнения внутреннего соединения таблиц. Если фрагментировать обе таблицы размера N по атрибутам соединения с помощью хеш-функции по модулю m, и затем выполнить соединение пофрагментно, то количество записей для обработки может составить не $N \cdot \ N$, а $\sum_1^m (\frac{N}{m})^2 = \frac{N^2}{m}$. Здесь мы предполагаем что хеш-функция обеспечивает равномерное распределение данных. В реальной ситуации, всё, конечно же может быть сложнее, однако выгода всё-равно присутствует.

Аналогично для Запроса 4 необходимо объединение таблиц Associate Professor и Full Professor и дальнейшее их соединение с University. Так как Associate Professor является одной из наиболее часто используемых таблиц, грамотное его фрагментирование способно уменьшить размеры промежуточных таблиц.

Запросы 8 и 10 обращаются к классу Student, то есть при выполнении потребуется объединение таблиц Graduate Student и Undergraduate Student и дальнейшее его соединение с таблицей Research Group для Запроса 8 и таблицей Teaching Course для запроса 10. Обе таблицы студентов являются как часто используемыми, так и одними из самых объемных в наборе данных. Фрагментирование этих таблиц способно существенно сократить количество записей для просмотра.

При горизонтальном фрагментировании запросы, интенсивно использующие иерархию из онтологии, могут пострадать. Однако этого не произойдет, так как система с полной материализацией, и при выполнении запроса сначала будет выполнен union фрагментов, а затем начнется выполнение. Если же отказаться от первичного объединения всех фрагментов, то иерархический запрос можно обрабатывать для каждого фрагмента отдельности, накапливая результаты и перебирая в цикле все фрагменты, пока накопленное множество не перестанет меняться.

Таким образом мы видим, что применение горизонтального фрагментирования к СУБД с выделением реляционной схемы в RDF данных может повысить производительность системы.

7.4. Реализация сбора статистики

Для осуществления сбора статистики было решено модифицировать структуру-дескриптор, с помощью которой хранится информация о характеристических множествах во время обработки запросов.

Каждое характеристическое множество было дополнено переменной-счетчиком $accessNumber$. В ходе исполнения запроса, выделяются звездчатые шаблоны и формируется граф соединений, в котором каждая

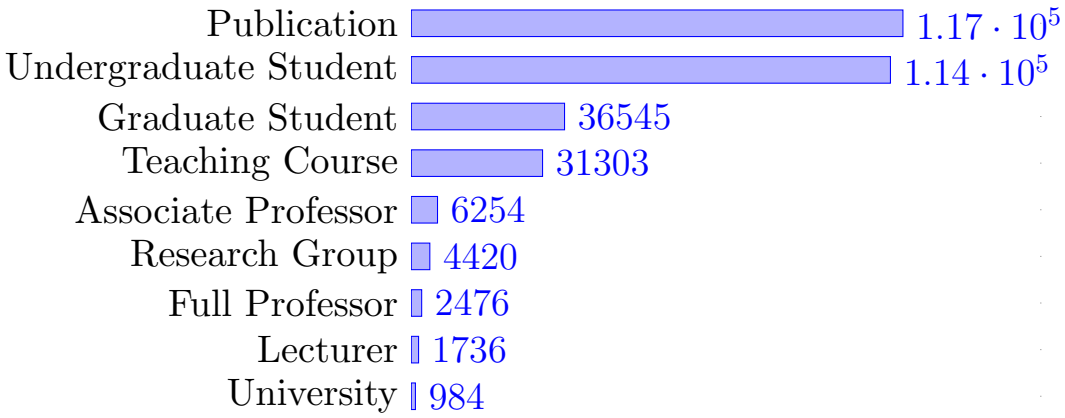

Рис. 4: Размеры Характеристических Множеств в записях

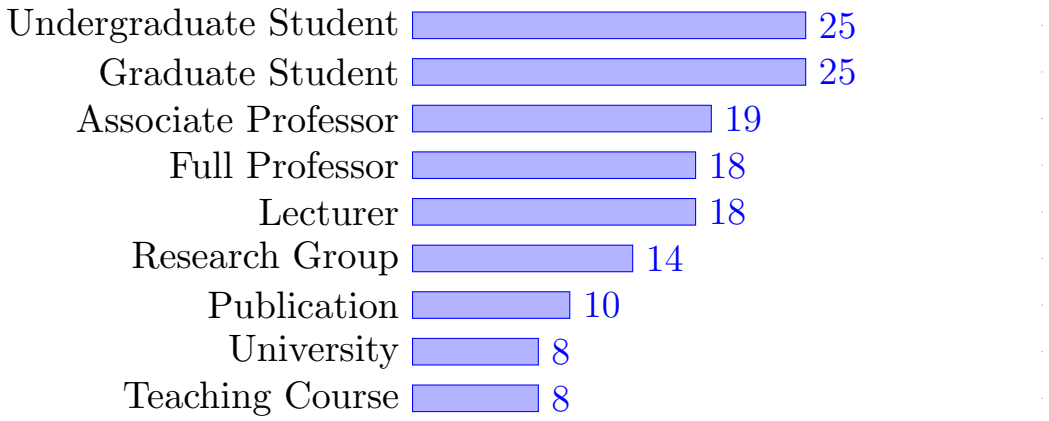

Рис. 5: Количество обращений к Характеристическим Множествам

	Запрос 2	Запрос 4	Запрос 8	Запрос 10
Publication				
Undergraduate Student			✓	✓
Graduate Student	✓		✓	✓
Teaching Course			✓	✓
Associate Professor		✓	✓	
Research Group	✓			
Full Professor		✓	✓	
Lecturer				
University	✓	✓	✓	

Таблица 2: Таблицы, обслуживающие запросы

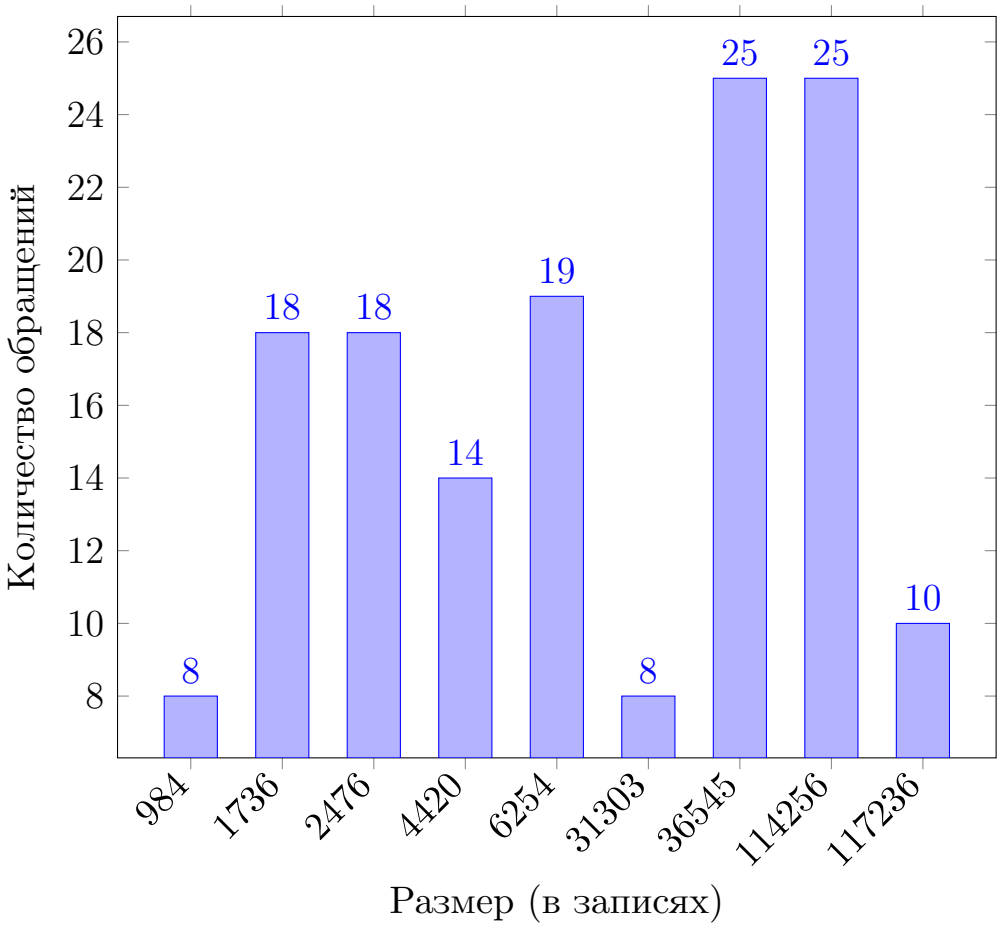

Рис. 6: Количество обращений относительно размера

таблица представлена своим идентификатором, назначаемым ей при выделении реляционной схемы. Как только все таблицы оказываются выбраны, для каждой из них увеличивается переменная $accessNumber$. При выборе таблиц для соединения с таблицами исключений в операторе $RDFScan$, происходит аналогичная процедура.

Чтобы выгрузить собранную информацию в файл, в MonetDB была создана хранимая процедура $rdf\_dumpStatistics()$. Она проходит по всем дескрипторам характеристических множеств, выводя в файл имя множества и значение $accessNumber$.

Информация о размерах характеристических множеств генерируется системой и выгружается в файл, если скомпилировать исходные коды с нулевым значением константы $NO\_OUTPUTFILE$.

Таблицы, обслуживающие запрос, выводятся в лог системы вместе с планом выполнения и другой отладочной информацией. По умолчанию эта функция отключена, для включения необходимо установить ненулевое значение константы $PRINT\_FOR\_DEBUG$.

8. Заключение

В ходе выполнения данной выпускной квалификационной работы было достигнуто следующее:

- произведен обзор предметной области и существующих решений проблемы;
- выбрана реализация выделения реляционной схемы в MonetDB/RDF;
- были реализованы фукции сбора статистики;
- выбранная реализация была ими инструментирована;
- проведены испытания;
- проанализирована полученная статистика и установлено распределение запросов по отношению к фрагментам;

По итогам анализа, применение горизонтального фрагментирования для эталонного набора тестов LUBM в системе MonetDB/RDF с выделением реляционной схемы имеет потенциал для улучшения производительности системы, а также может обеспечить более равномерное распределение нагрузки на данные.

9. Благодарности

Автор данной работы выражает благодарность автору исходного алгоритма выделения реляционной схемы Мин Дук Фаму за оперативные консультации и помощь в сборке системы.

Список литературы

[1] Bellatreche Ladjel, Woameno Komla Yamavo. Dimension Table Driven Approach to Referential Partition Relational Data Warehouses // Proceedings of the ACM Twelfth International Workshop on Data Warehousing and OLAP. — DOLAP '09. — New York, NY, USA : ACM, 2009. — P. 9–16. — URL: http://doi.acm.org/10.1145/1651291.1651294.

[2] Bizer Christian, Schultz Andreas. — Hershey, PA, USA : IGI Global, 2011. — P. 81–103. — ISBN: 9781609605933. — URL: http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-60960-593-3.ch004.

[3] DBpedia SPARQL Benchmark – Performance Assessment with Real Queries on Real Data / Mohamed Morsey, Jens Lehmann, Sören Auer, Axel-Cyrille Ngonga Ngomo // The Semantic Web – ISWC 2011: 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I / Ed. by Lora Aroyo, Chris Welty, Harith Alani et al. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2011. — P. 454–469. — ISBN: 978-3-642-25073-6. — URL: http://dx.doi.org/10.1007/978-3-642-25073-6_29.

[4] Deriving an Emergent Relational Schema from RDF Data / Minh-Duc Pham, Linnea Passing, Orri Erling, Peter Boncz // Proceedings of the 24th International Conference on World Wide Web. — WWW '15. — Republic and Canton of Geneva, Switzerland : International World Wide Web Conferences Steering Committee, 2015. — P. 864–874. — URL: https://doi.org/10.1145/2736277.2741121.

[5] Erling Orri. Virtuoso, a Hybrid RDBMS/Graph Column Store // IEEE Data Eng. Bull. — 2012. — Vol. 35, no. 1. — P. 3–8.

[6] Guo Yuanbo, Pan Zhengxiang, Heflin Jeff. LUBM: A benchmark for {OWL} knowledge base systems // Web Semantics: Science, Services and Agents on the World Wide Web. — 2005. — Vol. 3, no. 2–3. —

P. 158 – 182. — Selected Papers from the International Semantic Web Conference, 2004 ISWC, 20043rd. International Semantic Web Conference, 2004. URL: http://www.sciencedirect.com/science/article/pii/S1570826805000132.

[7] High Performance Parallel Database Processing and Grid Databases / David Taniar, Clement H. C. Leung, Wenny Rahayu, Sushant Goel. — Wiley Publishing, 2008. — ISBN: 0470107626, 9780470107621.

[8] Huang Jiewen, Abadi Daniel J., Ren Kun. Scalable SPARQL Querying of Large RDF Graphs // PVLDB. — 2011. — Vol. 4, no. 11. — P. 1123–1134. — URL: http://www.vldb.org/pvldb/vol4/p1123-huang.pdf.

[9] Klyne Graham, Carroll Jeremy J. Resource Description Framework (RDF): Concepts and Abstract Syntax // W3C Recommendation, W3C. — 2004. — URL: http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ (online; accessed: 2017-05-02).

[10] The LDBC Social Network Benchmark: Interactive Workload / Orri Erling, Alex Averbuch, Josep Larriba-Pey et al. // Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. — SIGMOD '15. — New York, NY, USA : ACM, 2015. — P. 619–630. — URL: http://doi.acm.org/10.1145/2723372.2742786.

[11] Mulay Kunal, Kumar P. Sreenivasa. SPOVC: A Scalable RDF Store Using Horizontal Partitioning and Column Oriented DBMS // Proceedings of the 4th International Workshop on Semantic Web Information Management. — SWIM '12. — New York, NY, USA : ACM, 2012. — P. 8:1–8:8. — URL: http://doi.acm.org/10.1145/2237867.2237875.

[12] Neumann T., Moerkotte G. Characteristic sets: Accurate cardinality estimation for RDF queries with multiple joins // 2011 IEEE 27th

International Conference on Data Engineering. — 2011. — April. — P. 984–994.

[13] Özsu M. Tamer, Valduriez Patrick. Principles of Distributed Database Systems (2Nd Ed.). — Upper Saddle River, NJ, USA : Prentice-Hall, Inc., 1999. — ISBN: 0-13-659707-6.

[14] Prud'hommeaux Eric, Seaborne Andy. SPARQL Query Language for RDF. — W3C Recommendation. — 2008. — URL: http://www.w3.org/TR/rdf-sparql-query/ (online; accessed: 2017-05-02).

[15] Query Workload-based RDF Graph Fragmentation and Allocation / Peng Peng, Lei Zou, Lei Chen, Dongyan Zhao // Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016. — 2016. — P. 377–388. — URL: http://dx.doi.org/10.5441/002/edbt.2016.35.

[16] Schroeder Rebeca, Hara Carmem S. Partitioning Templates for RDF // Advances in Databases and Information Systems: 19th East European Conference, ADBIS 2015, Poitiers, France, September 8-11, 2015, Proceedings / Ed. by Morzy Tadeusz, Patrick Valduriez, Ladjel Bellatreche. — Cham : Springer International Publishing, 2015. — P. 305–319. — ISBN: 978-3-319-23135-8. — URL: http://dx.doi.org/10.1007/978-3-319-23135-8_21.

[17] A distributed graph engine for web scale RDF data / Kai Zeng, Jiacheng Yang, Haixun Wang et al. // Proceedings of the 39th international conference on Very Large Data Bases. — PVLDB'13. — VLDB Endowment, 2013. — P. 265–276. — URL: http://dl.acm.org/citation.cfm?id=2488329.2488333.

[18] Г.А. Чернышев. Обзор подходов к организации физического уровня в СУБД // Тр. СПИИРАН. — 2013. — Vol. 24. — P. 222–276. — URL: http://mi.mathnet.ru/trspy580.