УДК 004.657

© 2017 г. **Ю.А. Григорьев,** д-р техн. наук,
В.А. Пролетарская
(Московский государственный технический университет им. Н.Э. Баумана),
Е.Ю. Ермаков, канд. техн. наук
(компания «Mail.ru Group»)

МЕТОД ДОСТУПА К ХРАНИЛИЩУ ДАННЫХ ПО ТЕХНОЛОГИИ SPARK С КАСКАДНЫМ ИСПОЛЬЗОВАНИЕМ ФИЛЬТРА БЛУМА*

Разработан новый метод выполнения SQL-запросов в среде параллельных вычислений Apache Spark. Он включает представление исходного запроса в виде нескольких подзапросов, разработку графа соединения и преобразования подзапросов, определение соединений, где необходимо использовать фильтры Блума, представление графа на языке Spark. На примере запроса Q3 теста TPC-H проведены натурные эксперименты, подтвердившие эффективность разработанного метода по сравнению с методом Hive.
Ключевые слова: запрос SQL, платформа Spark, фильтр Блума, тест TPC-H, схема «снежинка», схема «звезда», Hive, SQLContext, время выполнения запроса, производительность.

DOI: 10.22250/isu.2017.51.3-14

Введение

Стек Hadoop [1], базирующийся на парадигме MapReduce, прочно завоевал высокие позиции в рейтингах использования распределенной обработки и анализа данных [2]. Однако существуют альтернативы, обеспечивающие некоторые важные преимущества по сравнению с типичной Hadoop-платформой и значительно упрощающие описание процесса обработки данных. Одной из них является фреймворк Spark [3, 4].

Spark – это масштабируемая платформа анализа данных, которая включает примитивы для вычислений в оперативной памяти и, следовательно, позволяет выполнять многие операции за меньшее время. Spark реализован на Scala и поддерживает этот язык, который обеспечивает среду для обработки данных. Более того, ядро Spark дополняется набором мощных высокоуровневых библиотек, которые просто стыкуются с ним в рамках того же приложения. Хотя Spark предна-

* Работа выполнена при финансовой поддержке РФФИ в рамках научного проекта 16-37-00117 “Комплекс поддержки принятия решения на этапе проектирования Хранилища данных”.

значен для решения итеративных задач с распределенными данными, он фактически дополняет Hadoop и может работать вместе с файловой системой Hadoop.

Концепция MapReduce/Spark создала целый набор новых парадигм и структур для создания и обработки различных типов запросов. В рамках данной концепции существует несколько методов доступа к хранилищу данных (ХД), которые различаются возможным количеством таблиц в запросе, количеством фаз, форматом хранения записей, распределением данных по узлам и использованием ранней или поздней материализации результатов поиска. Критический анализ существующих методов был выполнен в [5 – 9]. В этих работах были также предложены методы, позволяющие устранить некоторые из существующих недостатков. Однако эти методы имеют ограниченные рамки применимости, а также прямую зависимость от объема оперативной памяти узла.

В статье предлагается метод доступа к хранилищу данных с произвольной схемой. Он основан на каскадном использовании фильтра Блума (КИФБ) и позволяет реализовать практически произвольный SQL-запрос в среде Spark.

Основные особенности метода с каскадным использованием фильтра Блума

В разработанном методе за основу взят метод SBFCJ – Spark Bloom-Filtered Cascade Join, предложенный в [6]. Учитывалось, что новый метод выполнения запроса к хранилищу данных должен быть применим и к хранилищам с произвольной структурой, а не только к хранилищам типа «звезда».

Этот метод основан на использовании широковещательных переменных Spark (broadcast) [10], которые в рамках данной технологии обладают значительной гибкостью и различаются по типу хранения, размеру блока и общей структуре.

Другой особенностью предложенного метода является использование фильтра Блума [11,12] – специально построенного массива битов на основе хеширования значений ключей таблиц. Этот фильтр дает ложноположительный результат с вероятностью $1/2^K$, где K – параметр фильтра Блума (число хеш-функций). При достаточно большом значении K можно добиться, чтобы вероятность ложноположительного срабатывания фильтра была бы очень небольшой. В таком случае уменьшается число записей, участвующих в соединении, а в некоторых случаях вообще не требуется выполнять соединение таблицы измерения с таблицей фактов. Это позволяет существенно уменьшить время обработки запросов.

На рис. 1 представлена общая схема использования фильтра Блума с целью уменьшения числа записей таблицы фактов, участвующих в соединении с таблицей измерения. Просматривается таблица измерения. Для записи, удовлетворяющей ограничениям, вычисляются K хеш-функций для ключа записи. Значение каждой хеш-функции – это смещение в битовом массиве. В соответствующей позиции массива устанавливается 1. После того, как все записи таблицы измерения обработаны, битовый массив с помощью широковещательной переменной broadcast автоматически тиражируется на все узлы, где хранятся блоки таблицы фактов. Далее просматриваются записи таблицы фактов. Для записи, удовлетворяю-

щей ограничениям, вычисляются K хеш-функций для внешнего ключа записи. И из соответствующих позиций массива читаются биты. Если все прочитанные K битов равны 1, то запись принимается для обработки. В противном случае она отсеивается.

Если значение внешнего ключа совпадает с одним из требуемых ключей таблицы измерения, то запись таблицы фактов всегда принимается на обработку (по построению фильтра Блума). Их число на рис. 1 обозначено как Q(+). Но возможно ложноположительное срабатывание фильтра. Их среднее число равно $Q1 = Q(–)/2^K$, Q(–) – это число записей таблицы фактов, значения ключей которых не удовлетворяют требованиям. Уменьшение Q1 позволяет существенно сократить объем данных, пересылаемых в процессе «перетасовки» (shuffle) при выполнении операции соединения (join) таблицы измерения и таблицы фактов.

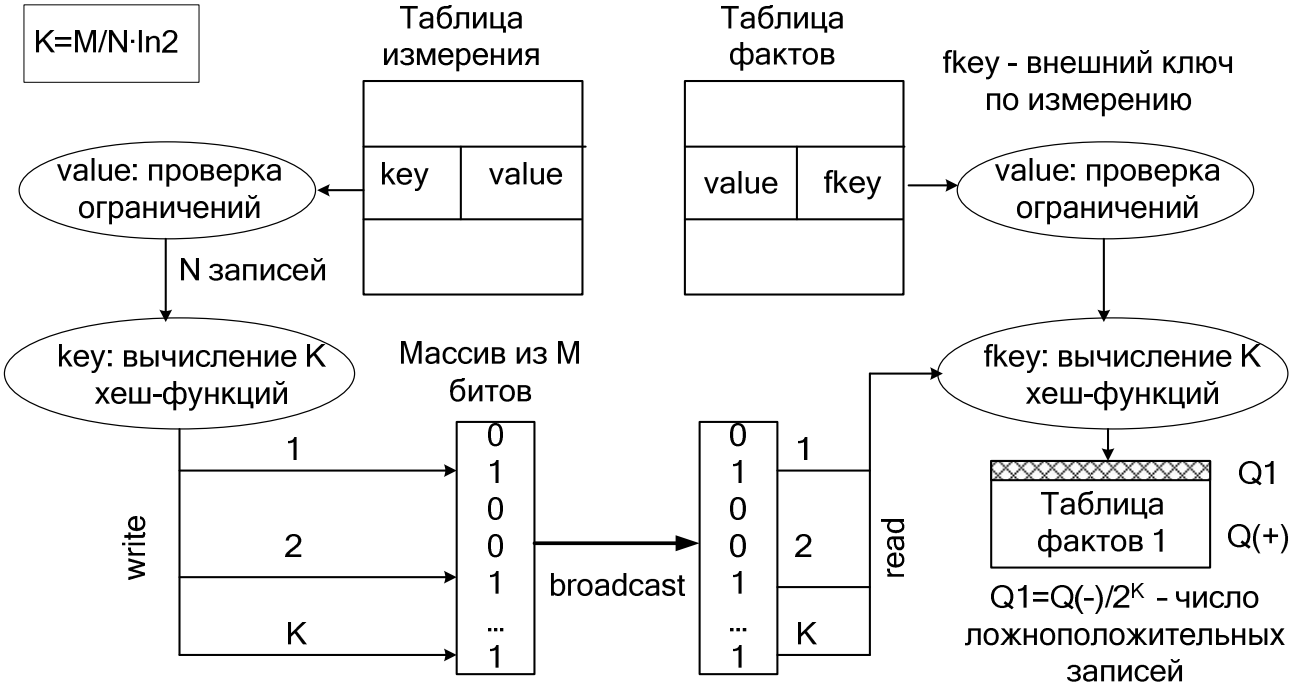

Рис. 1. Организация работы фильтра Блума.

Следует отметить, что количество хеш-функций $K = M/N \cdot \ln 2$ является оптимальным значением, минимизирующим вероятность ложноположительного срабатывания фильтра Блума для заданных M и N, где M – длина битового массива; N – число записей таблицы измерения, удовлетворяющих заданным ограничениям.

Фильтр Блума в рамках предложенного метода предлагается использовать в виде каскада, т.е. при каждом последующем соединении таблиц по новому измерению будет строиться новый фильтр Блума, что в значительной мере ускоряет выполнение операции join.

Основные принципы работы SPARK

Ключевой единицей работы с данными для Spark является RDD (Resilient Distributed Dataset), который представляет собой надежную распределенную таб-

лицу (но может быть и произвольным набором) [3]. RDD может быть полностью виртуальной и просто иметь инструкции по своему созданию, чтобы в случае сбоя узла иметь возможность для восстановления. Внутри RDD разбита на партиции (разделы), являющиеся минимальным объемом RDD, который будет обработан каждым рабочим узлом.

Пользователи могут создавать RDD двумя способами: загружая внешние наборы данных или распределяя коллекции объектов (например, списки или множества) внутри программы-драйвера. После создания RDD появляется возможность выполнять два вида операций: преобразования (transformations) и действия (actions). Преобразования создают новые наборы RDD на основе существующих. Примером типичного преобразования может служить фильтрация данных по заданному условию. Действия, напротив, вычисляют результат, не создавая новых наборов RDD, и возвращают его программе-драйверу или сохраняют во внешнем хранилище (например, в HDFS).

Реализация метода КИФБ в SPARK

Разработанный метод включает следующие шаги:

1) представление исходного запроса в виде нескольких подзапросов; каждый подзапрос выполняет поиск в одной таблице;

2) разработка графа соединения и преобразования подзапросов, а также определение соединений, где необходимо использовать фильтры Блума;

3) представление графа на языке Spark.

Проиллюстрируем этот метод на конкретном примере. Возьмем достаточно сложный запрос Q17 из теста TCP-H [13], включающий коррелированный подзапрос (табл. 1, столбец 1). Этот запрос определяет, насколько в среднем будет потерян годовой доход, если заказы не были реализованы для небольших количеств определенных партий.

Таблица 1

Запрос Q17	Подзапросы
select sum(l_extendedprice)/7.0 as avg_yearly **from** lineitem, part **where** p_partkey = l_partkey and p_brand = '[BRAND]' and p_container = '[CONTAINER]' and l_quantity < (**select** 0.2 * avg(l_quantity) **from** lineitem **where** l_partkey = p_partkey);	*Подзапрос P1:* **select** p_partkey as pr1 **from** part **where** p_brand = '[BRAND]' and p_container = '[CONTAINER]'; *Подзапрос L1:* **select** l_partkey as pr1, 0.2 * avg(l_quantity) as a1 **from** lineitem **where** l_partkey=BF1 **group by** l_partkey; *Подзапрос L2:* **select** l_partkey as pr1, l_quantity as q1, l_extendedprice as e1 **from** lineitem **where** l_partkey=BF2;

Соответствующая схема базы данных приведена на рис. 2. Схема базы на рис. 2 – это ориентированный (связи 1: M), ациклический граф.

На основании описания запроса Q17 сформируем подзапросы P1, L1, L2, представленые в табл. 1 (столбец 2).

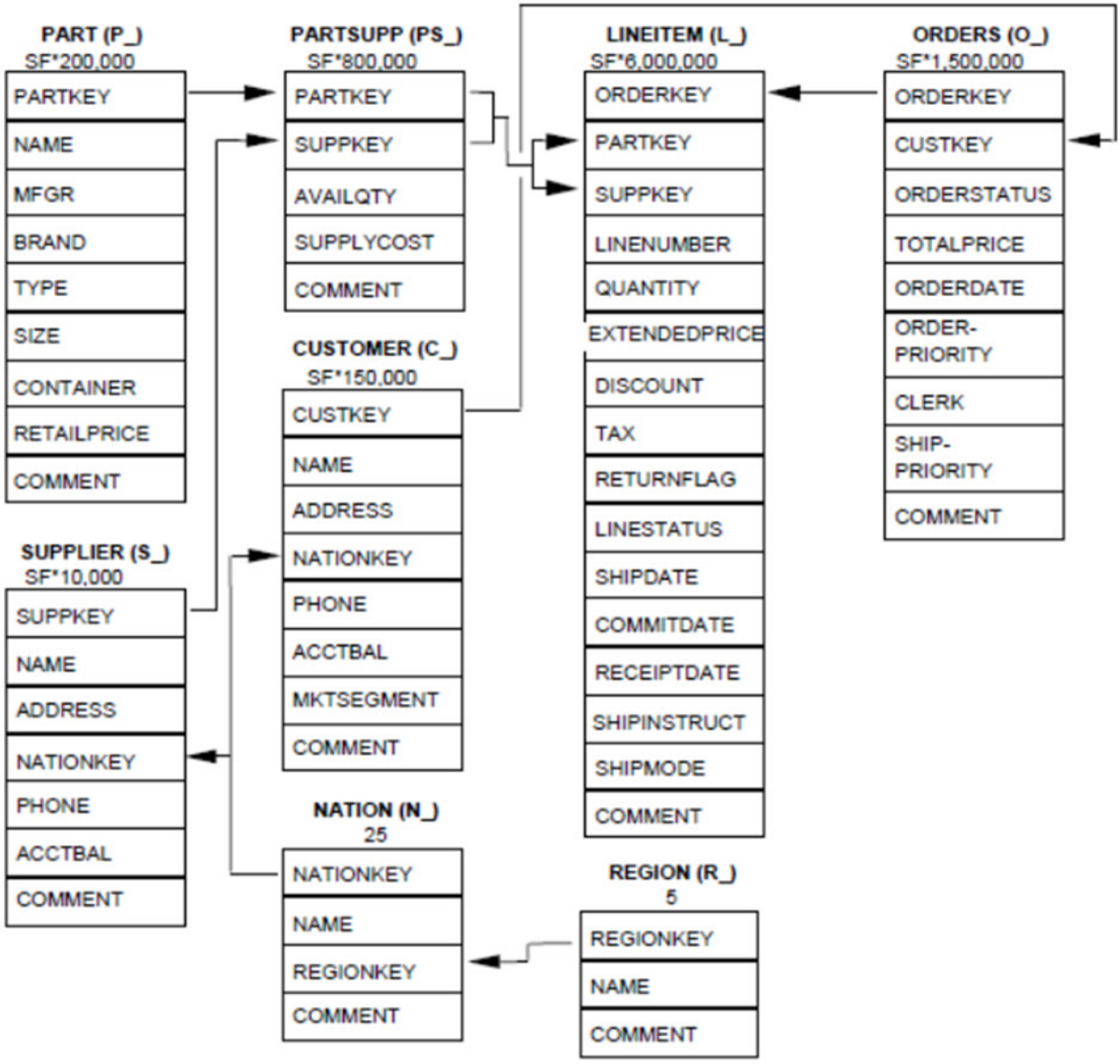

Рис. 2. Схема базы данных из теста TPC-H.

На рис. 3 приведен граф соединения и преобразования подзапросов. Здесь приняты следующие обозначения: прямоугольники обозначают исходные таблицы; прямоугольники с закругленными углами – таблицы (RDD), которые формируются Spark после выполнения подзапросов P1, L1, L2; прямоугольники с пунктирной границей – результаты соединения таблиц подзапросов; овалы – дополнительные операции (фильтрация, агрегирование). Дуги со стрелками обозначают исходные данные для формирования таблиц или выполнения операций, дуги без стрелок – условия соединений соответствующих таблиц. Кружками обозначены фильтры Блума (над ними указаны атрибуты, для которых строятся эти фильтры). Выполнение подзапроса P1 позволяет уменьшить мощность атрибута pr1. Поэтому использование здесь фильтров Блума BF1 и BF2 целесообразно. На рис. 4 приведены спецификации программы для Spark.

Из рис. 3 и 4 видно, что здесь имело место каскадное использование фильтра Блума (см. BF1 и BF2).

В приведенном выше примере соединяемые таблицы образуют «снежинку» с одним измерением в каждой «звезде» (рис. 3): P1-L1 (измерение P1, таблица фактов L1) и P1L1-L2 (измерение P1L1, таблица фактов L2). Из описания последовательности выполнения исходного запроса Q17 становится понятно, что по сети пересылаются компактные фильтры Блума, а не таблицы измерений. При этом соединение с таблицами фактов выполняется уже после применения к ним фильтров Блума, т.е. с уменьшенными по объему таблицами.

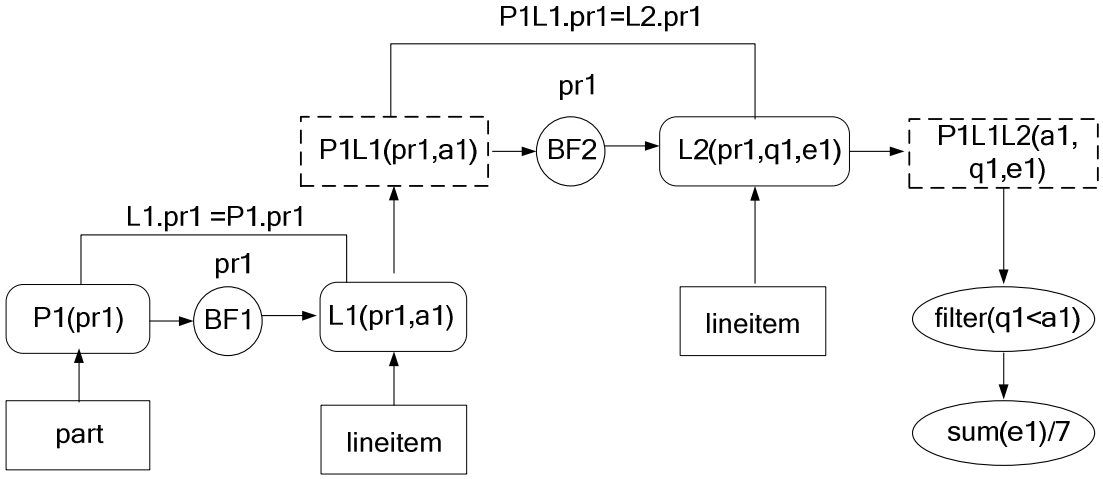

Рис. 3. Граф соединения и преобразования подзапросов для Q17.

 # Подзапрос P1
1. RDD1= part
2. RDD2=RDD1.filter(p_brand = '[BRAND]' and p_container = '[CONTAINER]').mapToPair(p_partkey as pr1,null) RDD2.persist(MEMORY_AND_DISK)
 # Построение фильтра Блума BF1
3.BF1 = broadcast(blum(RDD2.collect( )))
 # Подзапрос L1
4.RDD3= lineitem
5.RDD4=RDD3.filter(BF1.contains(l_partkey)). mapToPair(l_partkey as pr1, l_quantity).reduceByKey(def(0.2*avg(l_quantity) as a1))
 # Формирование таблицы P1L1 (соединение таблиц P1 и L1)
6. RDD5 = RDD4.join(RDD2)
RDD5.persist (MEMORY_AND_DISK)
 # Построение фильтра Блума BF2
7.BF2 = broadcast(blum(RDD5.collect( )))
 # Подзапрос L2
8.RDD6= lineitem
9.RDD7=RDD6.filter(BF2.contains(l_partkey)).mapToPair(l_partkey as pr1, [l_quantity as q1, l_extendedprice as e1])
 # Формирование таблицы P1L1L2 (соединение таблиц P1L1 и L2),
 # фильтрация и агрегирование
10. Result = RDD7.join(RDD5).map([a1,q1,e1]).filter(q1<a1). reduce(def(sum(e1)/7.0 as avg_yearly))

Рис. 4. Спецификации программы для Spark, реализующей запрос Q17.

В общем случае в каждой «звезде» могут присутствовать несколько таблиц. При этом исходные таблицы могут повторяться в одном или разных узлах «снежинки». Например, таблица lineitem в запросе Q17 повторяется в двух узлах (см. табл. 1, столбец 1).

Последовательность преобразований и действий в Spark образует план DAG (Directed Acyclic Graph) выполнения запроса. В общем случае план может быть сложным и иметь вид ациклического графа. Он работает как конвейер, параллельно «проталкивая» разделы (партиции) RDD (фрагменты таблиц, которые хранятся в узлах кластера) между узлами графа. На рис. 5 приведен план для схемы «звезда», которая может выступать в качестве измерения в другой «звезде».

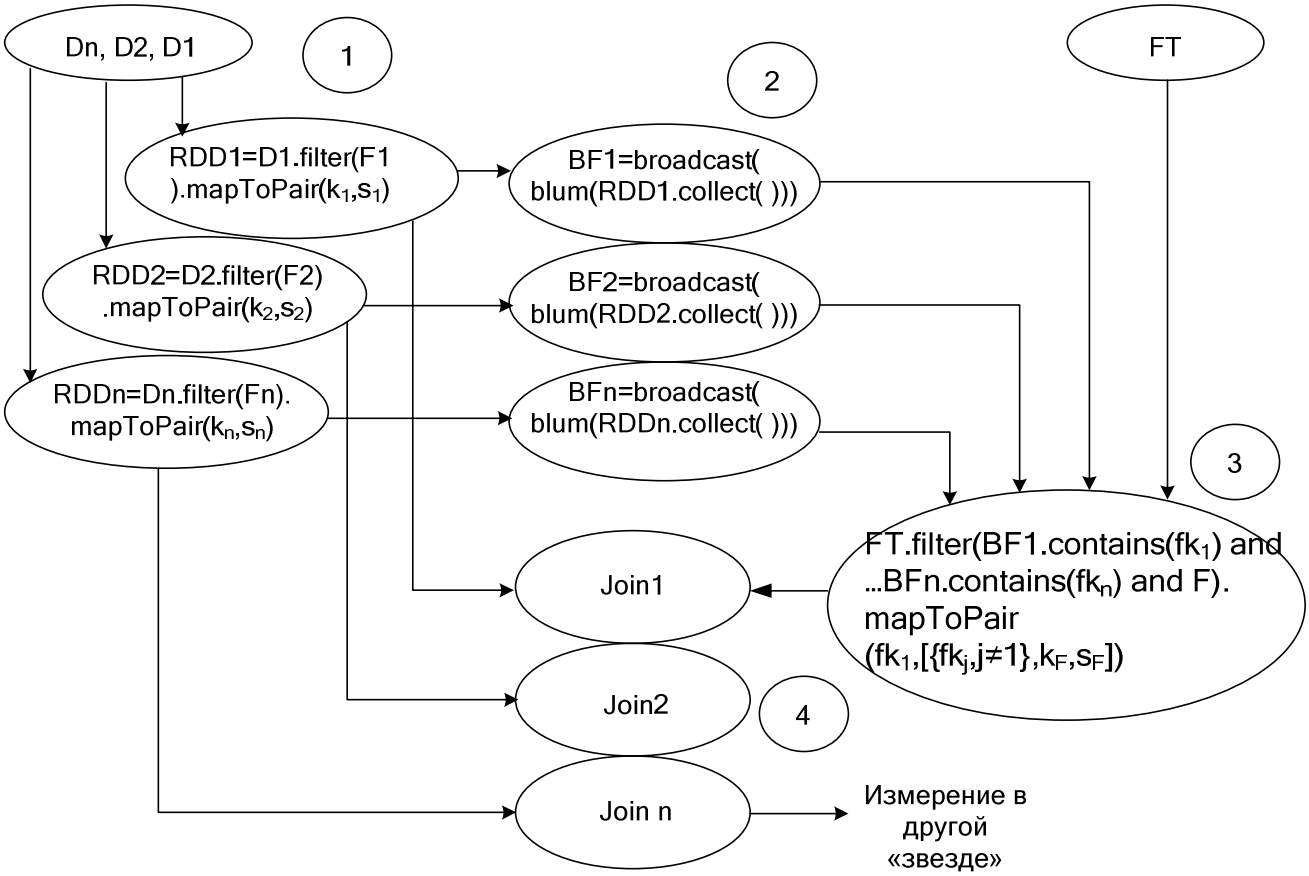

Рис. 5. Граф DAG.

Трек продвижения разделов имеет следующий вид (на рис. 5 цифрами обозначены номера этапов).

Этап 1. Чтение таблиц изменений D_i, фильтрация записей, получение проекции (k_i, s_i).

Этап 2. Сохранение ключей k_i на локальном диске (перед shuffle), сбор ключей измерений на одном Исполнителе (для каждого измерения), построение фильтров Блума (для каждого измерения), широковещательная рассылка фильтров Блума по всем Исполнителям, где выполняется фильтрация таблицы фактов (FT). Это точка синхронизации всех предыдущих процессов.

Этап 3. Чтение таблицы фактов, фильтрация записей со условию F и с помощью фильтров Блума, получение проекции ($\{fk_i\}$, k_F, s_F),

Этап 4. Последовательное соединение отфильтрованной таблицы фактов с отфильтрованными таблицами измерений (Join i). Группирование и сортировка, если это предусмотрено для данной «звезды». Переход к следующей «звезде».

Описание запроса, используемого в натурном эксперименте

Чтобы можно было сравнить разработанный метод КИБФ с другими методами, был выбран запрос Q3 (без коррелированного подзапроса) теста TCP-H [13]. В табл. 2 представлен сам запрос Q3 и приведено описание его подзапросов.

Таблица 2

Запрос Q3	Подзапросы

select l_orderkey,
sum(l_extendedprice*(1-l_discount)) as
revenue, o_orderdate, o_shippriority
from customer C_, orders O_,
lineitem L_
where c_mktsegment = '[SEGMENT]'
and c_custkey = o_custkey
and l_orderkey = o_orderkey
and o_orderdate < date '[DATE]'
and l_shipdate > date '[DATE]'
group by l_orderkey, o_orderdate,
o_shippriority
order by revenue desc, o_orderdate;
	Подзапрос C1:
	select c_custkey as kc1
	from C_
	where c_mktsegment = '[SEGMENT]';
	Подзапрос O1:
	select o_custkey as kc1, o_orderkey as ko1,
	o_orderdate as dt1, o_shippriority as sh1
	from O_
	where o_orderdate < date '[DATE]'
	and o_custkey=BF1;
	Подзапрос L1:
	select l_orderkey as ko1, l_discount as d1,
	l_extendedprice as e1
	from L_
	where l_shipdate > date '[DATE]'
	and l_orderkey=BF2;

На рис. 6 приведен граф соединения и преобразования подзапросов для Q3, построенный по правилам, принятым в предыдущем разделе.

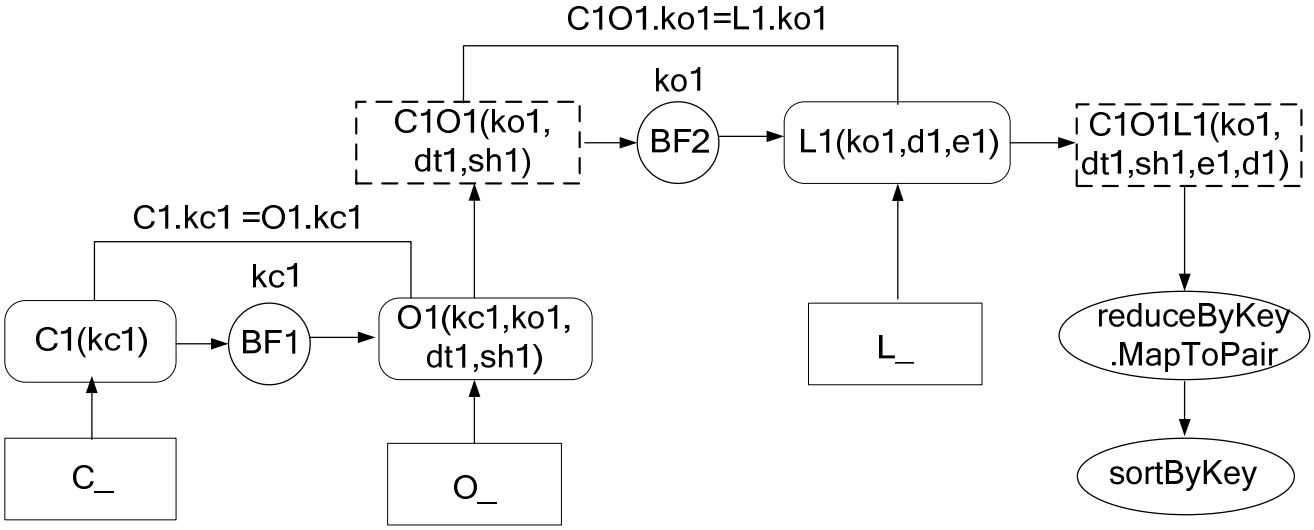

Рис. 6. Граф соединения и преобразования подзапросов для Q3.

На рис. 7 приведены спецификации программы для Spark, реализующей запрос Q3.

```
# Подзапрос C1
1. RDD1= C_
2.RDD2=RDD1.filter(c_mktsegment = '[SEGMENT]').mapToPair(c_custkey,null)
RDD2.persist(MEMORY_AND_DISK)
# Построение фильтра Блума BF1
3.BF1 = broadcast(blum(RDD2.collect( )))
# Подзапрос O1
4.RDD3= O_
5.RDD4=RDD3.filter(BF1.contains(o_custkey) and  o_orderdate < '[DATE]'). mapTo-
Pair(o_custkey, [o_orderkey, o_orderdate, o_shippriority])
# Формирование таблицы C1O1 (соединение таблиц C1 и O1)
6. RDD5 = RDD4.join( RDD2, numPartitions=m1).mapToPair (o_orderkey,[o_orderdate,
o_shippriority]);  RDD5.persist (MEMORY_AND_DISK)
# Построение фильтра Блума BF2
7.BF2 = broadcast(blum(RDD5.collect( )))
# Подзапрос L1
8. RDD7= L_
9.RDD8=RDD7.filter(BF2.contains(l_orderkey) and l_shipdate > '[DATE]'). mapTo-
Pair(l_orderkey, [l_extendedprice, l_discount] )
 # Формирование таблицы C1O1L1 (соединение таблиц C1O1 и L1)
10. RDD9 = RDD8 .join( RDD5,  numPartitions=m2).mapToPair([l_orderkey,
o_orderdate,o_shippriority], [l_extendedprice, l_discount])
# Группирование и сортировка записей
11. Result = RDD9.reduceByKey(def(sum(-l_extendedprice*(1-l_discount)) as revenue), num-
Partitions=num3).mapToPair([revenue, o_orderdate],[ l_orderkey,
o_shippriority]).sortByKey( )
```

Рис. 7. Спецификации программы для Spark, реализующей запрос Q3.

Результаты натурных экспериментов

Запрос Q3 запускался в четырех вариантах (т.е. для четырех методов) через консоль:

Hive (Hive) [1,3];

Pyspark с помощью стандартного SQLContext (Spark) [3];

Pyspark с каскадным использованием фильтра Блума (КИФБ);

Pyspark без использования фильтра Блума (БИФБ).

Каждый запрос запускался каждым методом не менее 6 раз.

Результаты экспериментов для одного узла приведены на рис. 8. Характеристики и узла были следующими: один двуядерный процессор, операционная система CentOS 7.2x64, оперативная память объемом 4Гб, два диска, каждый объемом 200 ГБ, объем базы данных 2(SF)x1 = 2 ГБ (SF – параметр теста TPC-H).

Из диаграммы видно, что метод КИБФ лучше Hive в 106/32 = 3,3 раза, но хуже SQLContext на (32 – 25)*100%/25 = 28%. Последнее связано с отсутствием фазы «перетасовки» (shuffle), так как эксперименты выполнялись на одном узле.

Далее для проведения экспериментального сравнения рассматриваемых методов был развернут виртуальный кластер. Кластер состоял из 8 узлов, на одном из которых были развернуты управляющие службы HDFS, Hive, Spark, Yarn.

Основные характеристики каждого виртуального узла были следующими: один двуядерный процессор, 8 GB оперативной памяти, 200 GB SSD диск, ОС Ubuntu 14.16.

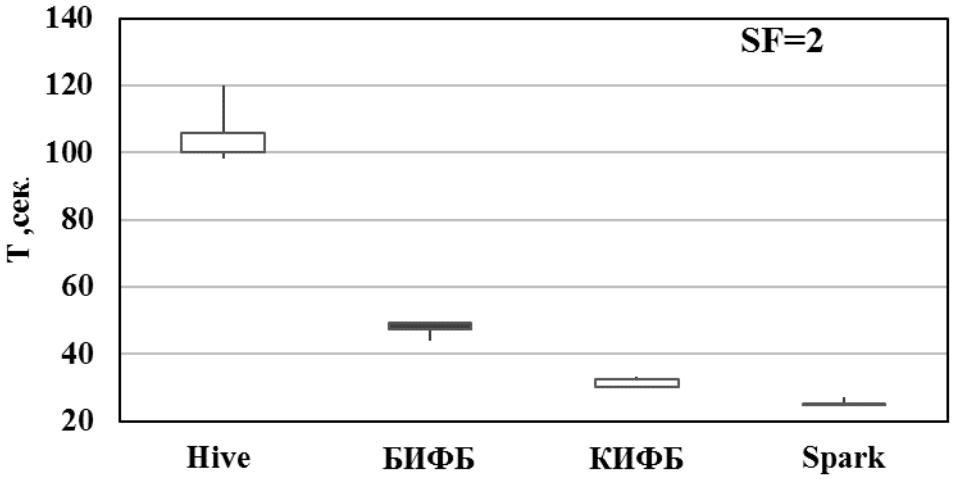

Рис. 8. Результаты экспериментов для одного узла.

На рис. 9 приведены результаты эксперимента для SF=500. Общий объем базы данных для этого случая составил около 500 ГБ.

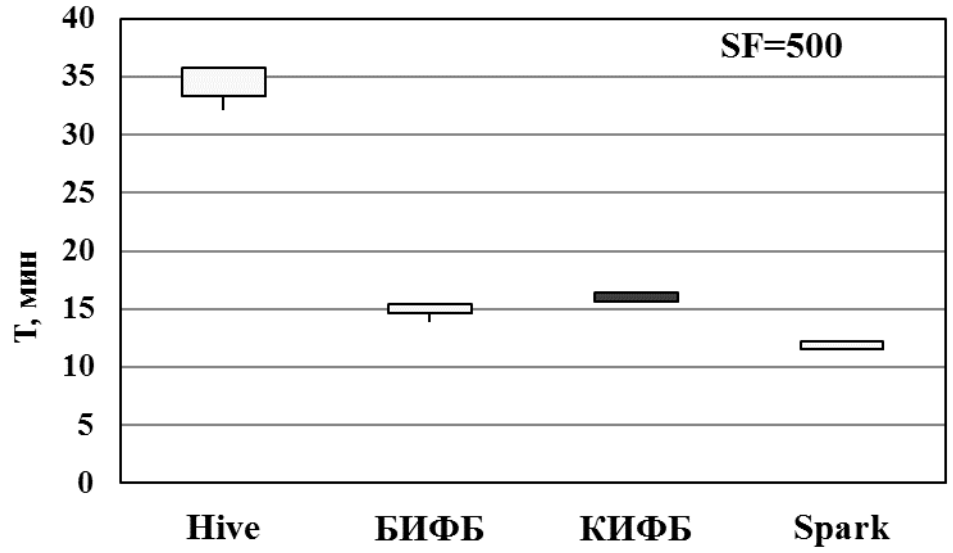

Рис. 9. Сравнение методов выполнения запроса Q3 для SF = 500.

Из диаграммы на рис. 9 видно, что метод КИБФ лучше Hive в 33,4/15,7 = 2,1 раза, но хуже SQLContext на (15,7 – 12,2)*100%/12,2 = 28%. Это связано с тем, что здесь «перетасовка» данных выполняется на уровне оперативной памяти, так как виртуальные узлы были развернуты на одном мощном физическом сервере. Поэтому эффект от использования фильтра Блума не проявился. В работе [6] использовались физические узлы и там эффект от использования фильтра Блума был намного выше. Для результатов, приведенных на рис. 9, большая часть времени приходится не на соединение таблиц, а на ввод/вывод промежуточных дан-

ных на диск из-за нехватки ОП (disk spill). Для варианта SQLContext это примерно 12,2 – 4,8 = 7,4 (мин.). Поэтому добавление лишней проверки (filter.mapToPair) в запрос значительно его «утяжеляет» (увеличивается disk spill). Для КИФБ это дополнительное «утяжеление» составляет 15,7 – 12,2 = 3,5 (мин.).

Заключение

Разработан метод выполнения запросов SQL с каскадным использованием фильтра Блума (КИФБ). Результаты натурных экспериментов на примере запроса Q3 подтвердили эффективность разработанного метода по сравнению с методом Hive. Для кластерной конфигурации предложенный метод значительно (в 2,1 раза) быстрее, чем Hive.

Хотя в проведенных экспериментах КИФБ и уступает SQLContext (на 28%) в среде виртуального кластера, существуют SQL-запросы, которые не могут быть реализованы в SQLContext, а с помощью КИФБ могут. Примерами таких запросов являются операторы Select с неравенством, которое накладывается на значение, возвращаемое коррелированным подзапросом (см. запрос Q17, табл. 1).

Предполагается продолжить эксперименты по выявлению преимуществ использования фильтра Блума для случая использования физического кластера.

ЛИТЕРАТУРА

1. *White T.* Hadoop: The Definitive Guide, 4th Edition. O'Reilly Media, 2015.
2. Аналитический обзор рынка BigData. [Электронный ресурс] [https://habrahabr.ru/company/moex/blog/256747/] Проверено 12.06.2016.
3. *Карау Х., Конвински Э., Венделл П., Захария М.* Изучаем Spark: молниеносный анализ данных. – М.: ДМК Пресс, 2015.
4. *Риза С., Лезерсон У., Оуэн Ш., Уиллс Д.* Spark для профессионалов: современные паттерны обработки больших данных. – СПб.: Питер, 2016.
5. *Ермаков Е.Ю., Пролетарская В.А.* Метод доступа к хранилищу данных по технологии MapReduce/Spark без кэширования таблиц измерений в оперативной памяти // Информационно-измерительные и управляющие системы. – 2016. – №12. – С. 90-97.
6. *Jaqueline Joice Brito, Thiago Mosqueiro, Ricardo Rodrigues Ciferri, Cristina Dutra de Aguiar Ciferri.* Faster cloud Star Joins with reduced disk spill and network communication. ICCS 2016. The International Conference on Computational Science. – 2016. – Vol. 80. – P. 74–85.
7. *Григорьев, Ю.А., Пролетарская В.А.* Метод ранней материализации доступа к хранилищу данных по технологии MapReduce // Информатика и системы управления. – 2015. – № 3. – С. 3-16.
8. *Григорьев Ю.А., Пролетарская В.А.* Сравнение методов обработки запросов к хранилищу данных по технологии MapReduce // Информатика и системы управления. – 2016. – № 1. – С. 3-13.
9. *G. Zhou, Y. Zhu, G.Wang.* Cache Conscious Star-Join in MapReduce Environments. Cloud-I '13 Proceedings of the 2nd International Workshop on Cloud Intelligence, August 26, 2013.
10. *Chowdhury M.* Performance and Scalability of Broadcast in Spark //2014-10-08]. http://people.eecs.berkeley.edu/~agearh/cs267.sp10/files/mosharaf-spark-bc-report-spring10.pdf. – 2014.
11. *Bloom B.H.* Space/time trade-offs in hash coding with allowable errors //Communications of the ACM. – 1970. – Т. 13, № 7. – С. 422-426.
12. *Григорьев Ю.А., Плутенко А.Д., Плужников В.Л., Ермаков Е.Ю., Цвященко Е.В., Пролетар-*

ская В.А. Теория и практика анализа параллельных систем баз данных. – Владивосток: Дальнаука, 2015.
13. Council T. P. P. TPC-H benchmark specification //Published at http://www.tpc.org/tpc_ documents_current_versions/pdf/tpc-h_v2.17.1.pdf. – 2017.

E-mail:
Григорьев Юрий Александрович – grigorev@bmstu.ru;
Ермаков Евгений Юрьевич – JK.Ermakov@gmail.com;
Пролетарская Виктория Андреевна – vilka2000@mail.ru.