УДК 004.65 DOI: 10.18698/0236-3933-2019-5-79-94

МОДЕЛИ ПРОЦЕССОВ СОЕДИНЕНИЯ ТАБЛИЦ ХРАНИЛИЩА ДАННЫХ ПО ТЕХНОЛОГИИ *MapReduce/Spark*

В.А. Пролетарская victoria.proletarskaya@gmail.com
Ю.А. Григорьев grigorev@bmstu.ru

МГТУ им. Н.Э. Баумана, Москва, Российская Федерация

Аннотация

Разработана модель и получена оценка передаваемого по сети объема данных при дублировании таблицы по узлам и с использованием фильтра Блума в среде *MapReduce*/*Spark*. Созданы модели процессов выполнения запросов на соединение таблиц базы данных при каскадном использовании фильтра Блума в этой же среде. Рассмотрены два случая соединения таблиц: 1) несколько кустов с одним измерением в каждом; 2) один куст с несколькими измерениями (хранилище типа «звезда»). Получена оценка объема фильтра Блума, передаваемого по сети при соединении таблиц. На примере запроса *Q3* из теста *TPC-H* выполнен анализ адекватности оценки выигрыша в объеме данных, передаваемых по сети при каскадном использовании фильтра Блума. Ошибка прогнозного значения составила 2 %

Ключевые слова

Big Data, MapReduce, Apache Spark, Spark SQL, фильтр Блума, TPC-H, модели процессов, хранилище данных

Поступила 28.05.2019

Введение. В последние несколько десятилетий в области обработки данных доминировали реляционные СУБД. В таких системах данные хранятся в виде таблиц, они также предполагают наличие схемы базы данных (БД). Однако при создании больших систем (*Big Data*) с использованием реляционных СУБД разработчики стали испытывать значительные затруднения. В частности, для хранении больших объемов информации необходимо покупать дорогостоящие специализированные аппаратно-программные комплексы параллельных систем баз данных (*Teradata*, *Sun Oracle Database Machine* и др.), а при наличии большого числа узлов возникает проблема обеспечения требуемой отказоустойчивости системы [1].

Для решения накопившихся проблем реляционных БД разработаны альтернативные средства хранения и обработки данных — базы данных *NoSQL* [2, 3]. Пионерами в этой области стали компании *Google* и

Amazon. В БД *NoSQL* для обеспечения высокой отказоустойчивости использована многократная репликация (копирование) записей. Такие БД имеют следующие недостатки [4]: в этих системах не поддерживается в полной мере режим ведения *ACID*-транзакций и блокировок, а также отсутствует *SQL*-интерфейс.

Для преодоления этих недостатков созданы БД *NewSQL* [5, 6]. Однако такие реляционные БД также имеют недостатки: наличие единой схемы базы данных; при выполнении запросов требуется чтение записей из большого числа связанных таблиц (проблема потери соответствия). Многие из этих БД являются коммерческими (имеют проприетарную лицензию).

Для хранения колоссальных объемов данных и их распределенной параллельной обработки используют специальные фреймворки, выполняющие запросы по технологии *MapReduce* [7]. Наибольшую популярность приобрели фреймворки *Hadoop* [8] и *Spark* [9–11]. В *Hadoop* прикладные программы (*Map*, *Reduce*) необходимо писать на языке низкого уровня *Java*. В *Spark* программы можно писать на языке высокого уровня *Scala* (можно и на *Python*), и в этом заключается основное его преимущество.

В настоящей работе рассмотрена проблема низкой производительности выполнения запросов в *Spark*, связанная с большим объемом данных, передаваемых по сети при выполнении соединения таблиц БД. Разработаны математические модели, с использованием которых можно оценить этот объем, а также получены решения, позволяющие уменьшить объем передаваемых по сети данных. В качестве СУБД для проведения натурного эксперимента использована *Apache Hive*.

Проблема производительности *Spark SQL*. *Spark* является экосистемой, включающей в себя несколько библиотек: *Spark SQL* — для реализации *SQL*-запросов, она интегрирована с *Hive*; *Spark Streaming* — для обработки потоков; *Spark GraphX* — для реализации вычислений с графами; *Spark MLib* — для машинного обучения. Здесь и далее речь идет об организации выполнения *SQL*-запросов в *Spark*, так как *SQL*-интерфейс доступа к данным является очень востребованным. Когда в прикладной программе (на языке *Scala*) встречается оператор *Select*, например соединение двух таблиц, то укрупненно его выполнение можно представить так, как показано на рис. 1.

На *Stage*0 сплиты (*Splits*) таблиц параллельно читаются в ОП на каждом узле. Сплит — фрагмент таблицы, хранящийся в узле (*Spark* стремится равномерно распределить сплиты таблицы по узлам). Далее *Spark* форми-

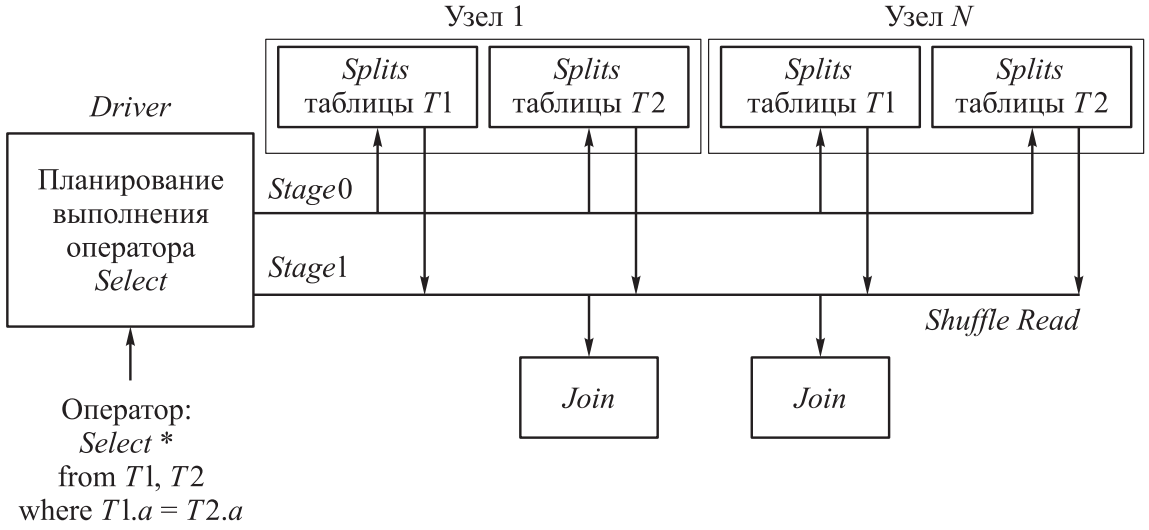

Рис. 1. Схема соединения таблиц *T*1 и *T*2 в *Spark SQL*

рует разделы (*Partitions*) таблиц на каждом узле. Записи с одним и тем же значением ключа «*a*» попадают в один раздел. На *Stage*1 разделы передаются по сети (*Shuffle Read*). Записи одного и того же раздела попадают в один узел *Join* (название условно), и там выполняется соединение записей таблиц *T*1 и *T*2.

На узлах *Join* задачи *Reduce* сортируют записи таблиц *T*1 и *T*2 по ключу и группируют их по значению ключа. Записи таблиц *T*1 и *T*2 с одним значением ключа попадают в группу (группы *a*1 и *a*3, рис. 2). Соединение сводится к построению декартова произведения записей таблиц *T*1 и *T*2, попавших в группу.

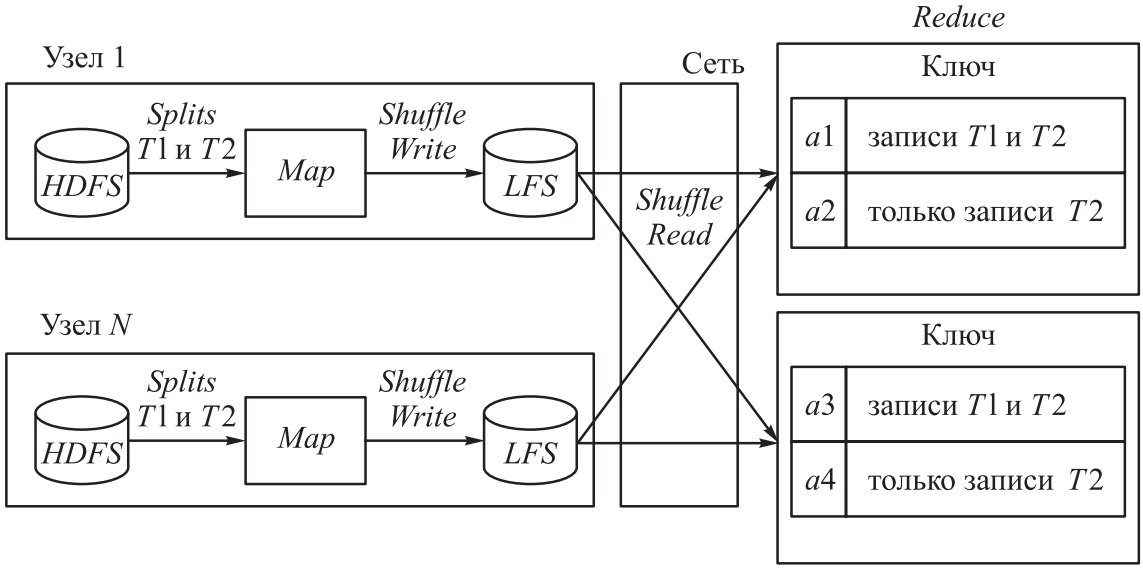

Рис. 2. Схема соединения записей таблиц *T*1 и *T*2 на узлах *Join*:
HDFS — распределенная файловая система; *LFS* — локальная файловая система

Возможны варианты, когда в группу попадают записи только одной таблицы, например $T2$ (группы $a2$ и $a4$, см. рис. 2), т. е. эти записи не имеют пары в таблице $T1$. Поэтому их обработка является бесполезной. Однако на это тратятся ресурсы системы: сохранение записей на локальном диске узла (*Shuffle Write*), чтение с диска и передача их по сети (*Shuffle Read*), обработка записей задачей *Reduce*. При большом размере таблицы $T2$ эти затраты становятся ощутимыми. Возникает проблема уменьшения числа бесполезных записей.

Дж. Брито (J.J. Brito) и другие [12] показали, что лучшими методами решения указанной проблемы являются два метода: 1) *SBFCJ* (*Spark Bloom-Filtered Cascade Join*) — использование фильтра Блума; 2) *SBJ* (*Spark Broadcast Join*) — дублирование таблиц. Эти методы лучше других по времени реакции (*Elapsed Time*), объему данных, пересылаемого между узлами (*Shuffled Bytes*), и объему данных, дополнительно сохраняемого на диске (*Disk Spill*). Сначала оценим передаваемые по сети объемы данных с использованием этих методов.

Оценка передаваемого по сети объема данных при дублировании таблицы (*SBJ*) и с использованием фильтра Блума (*SBFCJ*). Определим условие, когда каждый из этих методов *SBJ* и *SBFCJ* имеет преимущество. Пусть таблицы $T1$ и $T2$ связаны отношением 1:M ($T1$ — таблица измерений; $T2$ — таблица фактов). Схема соединения таблиц $T1$ и $T2$ с использованием дублирования таблицы $T1$ по всем узлам, где хранятся разделы таблицы $T2$, приведена на рис. 3.

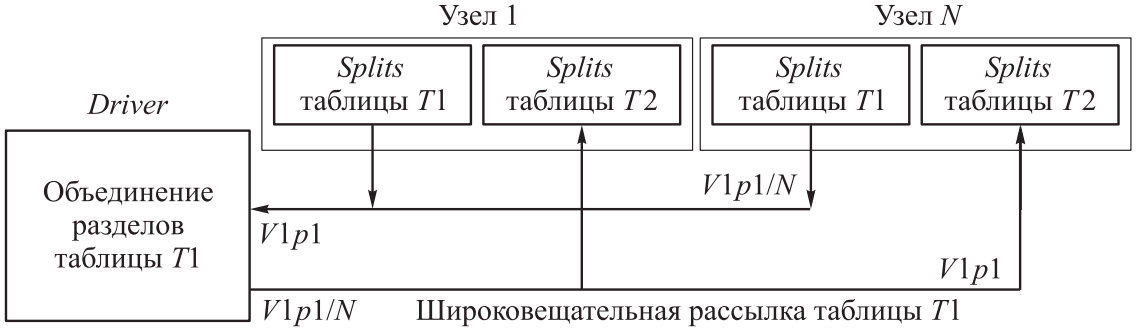

Рис. 3. Схема соединения таблиц $T1$ и $T2$ с помощью дублирования таблицы $T1$: N — число узлов; $V1$, $V2$ — объемы (в байтах) тех колонок таблиц $T1$ и $T2$, которые указаны в запросе *SELECT*; $p1$ — эффективная селективность (доля записей таблицы $T1$, удовлетворяющих условию подзапроса по этой таблице)

После фильтрации ($p1$) разделы таблицы $T1$ пересылаются на управляющий узел *Driver*. Там они объединяются (*union*), и уже вся отфильтрованная таблица $T1$ передается на узлы, где хранятся разделы таблицы

*T*2. На этих узлах выполняется соединение. Далее предполагается, что данные равномерно распределены по узлам (*Spark* стремится это реализовать). Объем данных, передаваемых по сети,

$$VD = p1V1(N+1). \quad (1)$$

Схема соединения таблиц *T*1 и *T*2 с помощью фильтра Блума приведена на рис. 4 [13, 14]. Для разделов таблицы *T*1 создаются фильтры Блума *BF*1 по атрибуту соединения (по первичному ключу — *PK*). Затем по команде *Collect* эти фильтры собираются на управляющем узле и объединяются по команде ИЛИ. Далее полученный фильтр пересылается на узлы, где хранятся разделы таблицы *T*2. Записи таблицы *T*2 дополнительно фильтруются с помощью полученного фильтра. После этого фрагменты таблиц *T*1 и *T*2 соединяются в узлах, где запущены задачи *Reduce*. Соединение необходимо, чтобы удалить лишние записи таблицы *T*2. Они могли появиться вследствие ложноположительного срабатывания фильтра Блума.

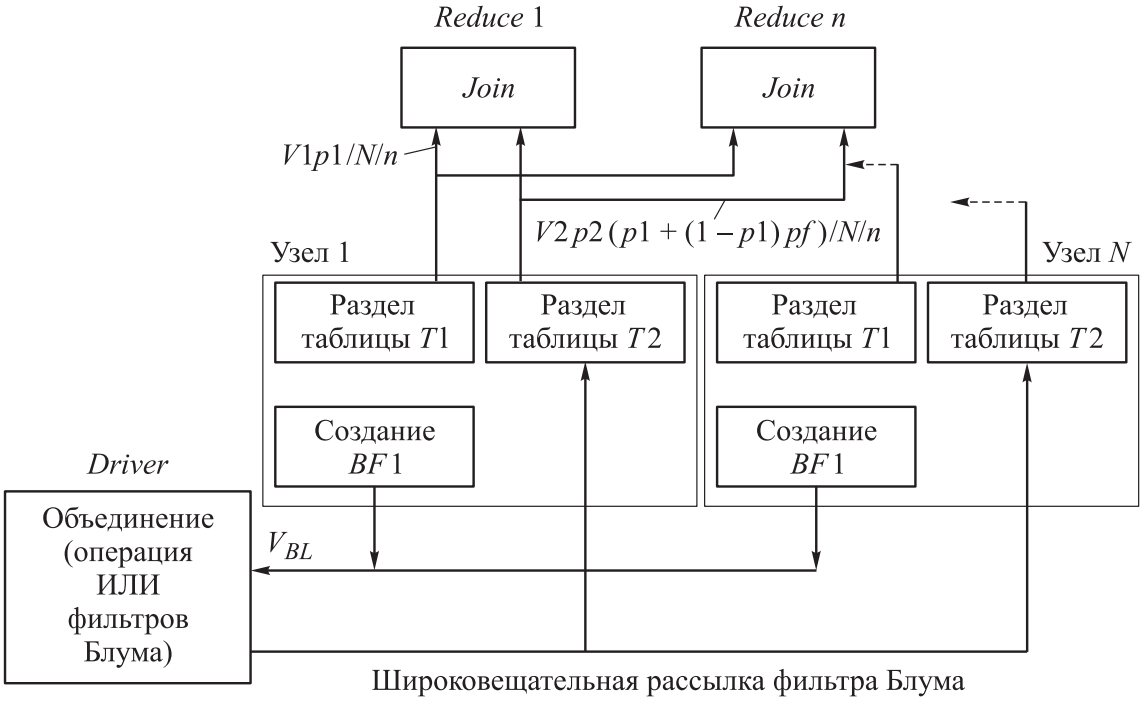

Рис. 4. Схема соединения таблиц *T*1 и *T*2 с помощью фильтра Блума:

V_{BL} — суммарный объем фильтров Блума, передаваемый по сети; *pf* — вероятность ложноположительного срабатывания фильтра Блума (в оптимальном случае равна $1/2^H$, где *H* — число хеш-функций); *n* — число задач (процессов) *Reduce*, где выполняется соединение фрагментов таблиц *T*1 и *T*2

Для некоторой таблицы *T* оценим объем данных, передаваемых по сети (*Shuffle*) в процессе ее соединения с другой таблицей. Пусть объем соединяемой таблицы *T* равен *V*. Тогда объем *Shuffle* равен

$$(V/N/n)nN = V. \quad (2)$$

Поясним формулу (2). Первый сомножитель (в скобках) — объем одного фрагмента таблицы на одном узле. Число фрагментов n равно числу задач *Reduce*. В фрагмент включаются записи, имеющие одинаковое значение хеш-функции для атрибута соединения z ($h(z) = i$, i — номер фрагмента). Второй сомножитель n учитывает число фрагментов одного узла. Третий сомножитель N показывает, что по сети распространяются фрагменты всех узлов.

Для отфильтрованной по условию поиска таблицы $T1$ (на это указывает вероятность $p1$) получим объем *Shuffle* (см. рис. 4):

$$V1p1. \quad (3)$$

Для отфильтрованной таблицы $T2$ (селекция по условию подзапроса — вероятность $p2$ и фильтр Блума $BF1$) объем *Shuffle* составляет

$$V2p2\,(p1\cdot 1 + (1 - p1)pf). \quad (4)$$

Поясним выражение в круглых скобках в (4). С вероятностью $p1$ запись таблицы $T2$ удовлетворяет условию соединения $T1.z = T2.z$, где z — первичный атрибут таблицы $T1$ (PK) и внешний ключ таблицы $T2$ (FK). Действительно, мощность атрибута z в таблице $T2$ равна $|T1|$, так как таблицы $T1$ и $T2$ связаны отношением 1:M. Число благоприятных значений z в $T2$ равно $p1\cdot|T1|$. Отсюда получим вероятность, что запись таблицы $T2$ удовлетворяет условию соединения: $p1\cdot|T1|/|T1| = p1$.

Выражение в круглых скобках формулы (4) — формула полной вероятности. По построению фильтра Блума он с вероятностью 1 пропускает запись, если она удовлетворяет условию соединения ($p1$). С вероятностью ложноположительного срабатывания pf фильтр Блума пропускает запись, если она не удовлетворяет условию соединения ($1 - p1$).

Вероятность $pf = 1/2^H$ в (4) можно принять равной нулю, если число H хеш-функций в фильтре Блума велико.

Суммируя (3) и (4), полагая $pf = 0$ и учитывая объем фильтра Блума, получаем формулу для оценки объема данных, передаваемых по сети в процессе соединения таблиц $T1$ и $T2$ с помощью фильтра Блума:

$$VB = V1p1 + V2p2p1 + V_{BL}. \quad (5)$$

Оценка объема V_{BL} фильтра Блума будет выполнена ниже.

Из (1) и (5) следует, что неравенство $VD < VB$ выполняется при

$$NV1 - V_{BL}/p1 < p2V2. \quad (6)$$

Таким образом, если выполняется неравенство (6), то дублирование отфильтрованной таблицы $T1$ имеет преимущество перед использованием фильтра Блума. В этом случае объем данных, передаваемых по сети в процессе соединения таблиц $T1$ и $T2$, меньше. В противном случае лучше использовать фильтр Блума.

Spark SQL поддерживает режим дублирования небольшой таблицы по узлам при выполнении операции соединения таблиц. Однако часто этот режим не выбирается оптимизатором *Spark SQL*, даже если он в соответствии с формулой (6) является рациональным.

Оценка передаваемого по сети объема данных при каскадном использовании фильтра Блума (КИФБ). Рассмотренный в работе [12] метод *SBFCJ* хорошо работает при больших значениях $V1$ и N (см. (6)) или при небольших объемах оперативной памяти (ОП). В этом методе фильтр Блума используется для реализации запросов только к хранилищу данных типа «звезда». В этом случае измерения хранятся в ненормализованных таблицах, и число их столбцов может большим [15, с. 127]. Кроме того, в одном запросе нельзя сочетать преимущества обоих методов (*SBJ* и *SBFCJ*).

Новый метод, основанный на каскадном использовании фильтра Блума (КИФБ, *Bloom Filter Cascade Application*, *BFCA*), предложен в работах [16–18]. Метод имеет следующие преимущества:

– возможность реализации *SQL*-запросов к хранилищу данных с произвольной схемой (типа «снежинка», измерения хранятся в нормализованных таблицах, т. е. разбиваются на подызмерения);

– возможность сочетания фильтра Блума и дублирования небольших промежуточных таблиц при выполнении одного запроса, включающего в себя операции соединения таблиц;

– возможность реализации операторов *Select*, которые не могут быть выполнены напрямую в среде *Spark SQL*.

Далее приведена оценка выигрыша в объеме передаваемых по сети данных при использовании метода КИФБ. Рассмотрим два случая.

Случай 1. K–1 кустов с одним измерением в каждом (рис. 5, *а*). Таблица T_{i1} отличается от T_i тем, что она может включать в себя колонки других таблиц как результат предыдущих соединений, но при этом $|T_{i1}| = |T_i|$. Предположим, что таблицы T_i и T_{i+1} связаны отношением 1:M, $i = 1, ..., K–1$. Чтобы не использовать конкретные числовые значения в качестве индексов, примем следующие обозначения эквивалентными: $T_1 \equiv T1$, $p_1 \equiv p1$, $V_1 \equiv V1$, $V_{21} \equiv V21$ и т. д.

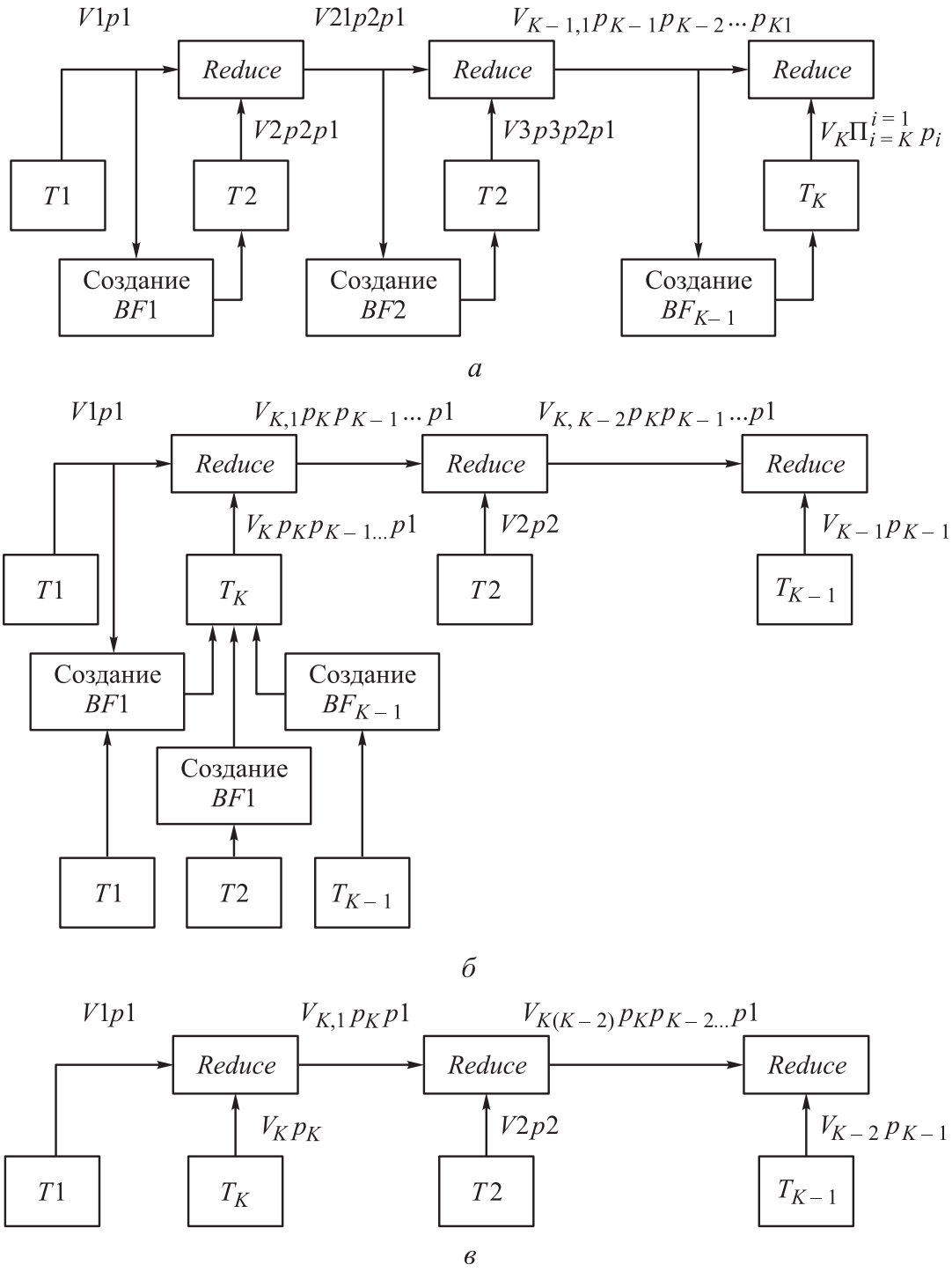

Рис. 5. Схемы КИФБ (*BFi* — фильтры Блума) (*а*), соединения *K*–1 таблиц измерений с таблицей фактов с использованием (*б*) и без использования (*в*) фильтров Блума

Предложенный метод КИФБ заключается в следующем (см. рис. 5, *а*). Создается фильтр Блума *BF*1 по первичному ключу *z*1 отфильтрованной таблицы *T*1 (обозначим ее как *T*1(*p*1)). Таблица *T*2(*p*2) дополнительно фильтруется по *BF*1 и соединяется с *T*1(*p*1), получается таблица *T*21(*p*2*p*1).

Далее создается новый фильтр Блума *BF*2 по первичному ключу *z*2 таблицы *T*21(*p*2*p*1). Предыдущий фильтр *BF*1 удаляется. Таблица *T*3(*p*3) дополнительно фильтруется по *BF*2 и соединяется с *T*21(*p*2*p*1) и т. д.

Рассмотрим соединение таблиц $T_{i1}(p_i p_{i-1}...p1)$ и $T_{i+1}(p_{i+1}p_i...p1)$, $i = 1, ..., K-1$, $T_{11}=T1$. По аналогии с (5) получим оценку объема *Shuffle* для метода КИФБ:

$$V_{i1}\left(\prod_{j=i}^{1} p_j\right) + V_{i+1}p_{i+1}\left(\prod_{j=i}^{1} p_j\right) + V_{BLi}. \qquad (7)$$

Если фильтры Блума не используются, то таблица $T_{i1}(p_i p_{i-1}...p1)$ соединяется с таблицей $T_{i+1}(p_{i+1})$. В этом случае оценка объема *Shuffle* имеет вид

$$V_{i1}\left(\prod_{j=i}^{1} p_j\right) + V_{i+1}p_{i+1}. \qquad (8)$$

Вычтем (7) из (8) и просуммируем по *i*. По методу КИФБ получим выигрыш в объеме данных, передаваемых по сети:

$$\Delta = \sum_{i=1}^{K-1}\left(V_{i+1}p_{i+1}\left(1-\prod_{j=i}^{1} p_j\right) - V_{BLi}\right). \qquad (9)$$

Пусть $p1$ мало и все записи таблиц, начиная с *T*2, читаются полностью ($p_i = 1$, $i = 2, ..., K$). Тогда эффект от применения метода КИФБ можно оценить как

$$\Delta = \sum_{i=1}^{K-1}\left(V_{i+1} - V_{BLi}\right). \qquad (10)$$

Это объем всех участвующих в соединении таблиц, начиная с *T*2, без объемов фильтров Блума, т. е. эффект может быть большим. Это важно, поскольку в задачах принятия решений часто в запрос включают много таблиц.

Случай 2. Один куст с *K*–1 измерениями (рис. 5, *б*). Этот случай соответствует реализации запроса к хранилищу данных типа «звезда» (метод *SBFCJ* [12]). Объем *Shuffle* при соединении таблиц измерений с таблицей фактов с использованием фильтров Блума (см. рис. 5, *б*):

$$\sum_{i=1}^{K-1} V_i p_i + \sum_{i=1}^{K-2} V_{Ki}\prod_{j=K}^{1} p_j + V_K\prod_{j=K}^{1} p_j + \sum_{i=1}^{K-1} VBLi,\ V_{Ki} > V_K. \qquad (11)$$

Объем *Shuffle* при выполнении соединений без фильтра Блума (рис. 5, *в*):

$$\sum_{i=1}^{K-1} V_i p_i + \sum_{i=1}^{K-2} V_{Ki} p_K \prod_{j=i}^{1} p_j + V_K p_K. \quad (12)$$

Вычитая (11) из (12), получаем выигрыш в объеме *Shuffle*

$$\Delta = \sum_{i=1}^{K-2} V_{Ki} p_K \prod_{j=i}^{1} p_j \left(1 - \prod_{j=K-1}^{i+1} p_j\right) + V_K p_K \left(1 - \prod_{j=K-1}^{1} p_1\right) - \sum_{i=1}^{K-1} V_{BLi}. \quad (13)$$

Пусть $p_i = p$, $i = 1, ..., K-1$, и $p_K = 1$. Тогда (13) можно переписать в виде

$$\Delta = \sum_{i=1}^{K-2} V_{Ki} \left(p^i - p^{K-1}\right) + V_K \left(1 - p^{K-1}\right) - \sum_{i=1}^{K-1} V_{BLi}. \quad (14)$$

Пусть p мало, а K достаточно велико и $V_{K\,i} = V_K$, тогда выигрыш

$$\Delta = V_K (1 + p) - \sum_{i=1}^{K-1} V_{BLi}. \quad (15)$$

Это объем, больший всей таблицы фактов (ее столбцов, участвующих в запросе), не содержит объемов фильтров Блума.

Оценка объема фильтра Блума, передаваемого по сети при выполнении соединения таблиц. Начиная с версии 2.3.0, в *Spark* для объединения фильтров Блума, созданных в узлах, используется метод *treeAggregate* (предложил O. Lojkine). Из текста *def treeAggregate* следует, что фильтры Блума создаются и затем объединяются в следующей последовательности:

1) всего на узлах выполняются *NS* задач (число *numPartitions* меньшей таблицы), созданные фильтры Блума сохраняются локально (*Shuffle Write*) на каждом узле;

2) создаются q = ceil(sqrt(*NS*)) — 1 задач, которые читают фильтры Блума (*Shuffle Read*) и объединяют их (операция ИЛИ);

3) объединенные фильтры передаются на узел *Driver* и там окончательно объединяются (операция ИЛИ). Далее конечный фильтр передается обратно на узлы (*N*), где хранятся разделы большей таблицы, и выполняется фильтрация записей этой таблицы. Из примера, рассмотренного в работе [13], следует, что фильтр Блума хорошо сжимается, коэффициент сжатия равен 48·90 МБ/1375 МБ = 3.

Пусть *NS* = 21, q = (ceil(sqrt(21)) — 1 = 4, *N* = 7 (рис. 6). Всего будет выполнено 20 перемещений фильтров Блума по сети: 1) *NS* — q·(*NS*/*N*) = = 9 — на узлы с q задачами; 2) q = 4 — на узел *Driver*; 3) *N* = 7 — обратно на семь рабочих станций.

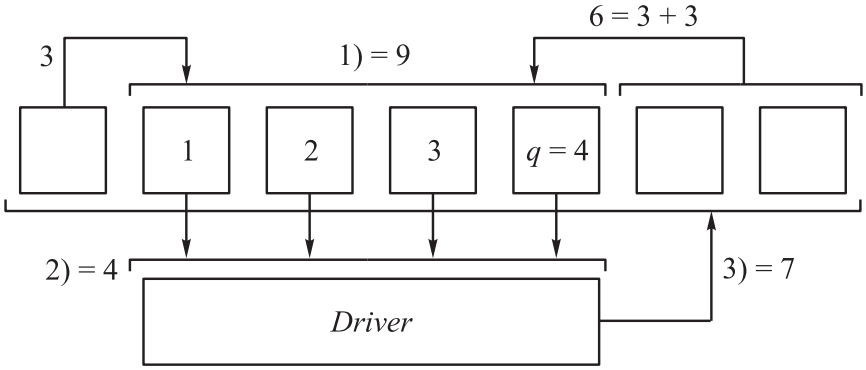

Рис. 6. Пример схемы перемещений фильтров Блума

Следовательно, формула для перемещаемого по сети объема фильтром Блума имеет вид

$$V_{BL} = \left(NS - q\left\lceil NS / N \right\rceil + q + N \right) QH / \ln 2 / 8 / 3, \qquad (16)$$

где q = ceil(sqrt(NS)) – 1; Q — предполагаемое число элементов в фильтре Блума; $H = \log_2(1/pf)$; pf — вероятность ложноположительного срабатывания фильтра Блума.

Поясним формулу (16). Выражение в скобках — число перемещений фильтров Блума по сети (см. приведенный выше пример), QH/ln2/8 — размер фильтра Блума в байтах [1]; 3 — коэффициент сжатия фильтра Блума.

Анализ адекватности оценки выигрыша в объеме данных, передаваемых по сети методом КИФБ. Для подтверждения адекватности оценки (9) использованы результаты натурных экспериментов, полученные на стенде. Стенд состоял из восьми узлов, на одном из которых инсталлированы управляющие службы *HDFS*, *Hive*, *Spark*, *Yarn* [21]. Основные характеристики каждого виртуального узла: один двухъядерный процессор, 8 ГБ оперативной памяти, 200 ГБ *SSD* диск, ОС *Ubuntu* 14.16. В качестве дистрибутива *Hadoop* использована *Cloudera*. Эксперименты проводились для запроса *Q*3 (соединение трех таблиц) из теста *TPC-H* [21]. Этот запрос реализован в среде *Spark SQL* без и с использованием метода КИФБ. Спецификации запроса приведены в работах [16, 17, 21]. Параметр наполнения базы данных SF = 500.

Стенд развернут в виртуальной среде. Объемы передаваемых данных и число записей зависят от значения параметра SF наполнения базы данных и не зависят от того, являются ли узлы виртуальными или физическими. Следует отметить, что проекции таблиц, передаваемые по *Shuffle* (V_i), не сжимаются по умолчанию, поэтому часто объем *Shuffle Write* превышает объем *Input*.

Для реализации запроса $Q3$ выполнены операции соединения: ($T1$►◄ $T2$)►◄$T3$, $T1$ — исходная таблица CUSTOMER (C_), $T2$ — исходная таблица ORDERS (O_), $T3$ — исходная таблица LINEITEM (L_).

Значения параметров для таблиц C_, O_, L_ запроса $Q3$ и рассчитанные по формуле (16) объемы фильтров Блума ($N = 7$) приведены в таблице.

Значения параметров для таблиц C_, O_, L_ запроса $Q3$ и рассчитанные по формуле (16) объемы фильтров Блума ($N = 7$)

Таблица	V, ГБ	p	NS	$Q/(pf)$	V_{BLi}, ГБ
$T1$(C_)	$V1 = 0{,}35$	$p1 = 0{,}20$	24	$15 \cdot 10^6$/ 0,01	$V_{BL1} = 0{,}11$
$T2$(O_)	$V2 = 9{,}0$	$p2 = 0{,}49$	–	–	–
$T1$►◄ $T2$	–	–	14	$40 \cdot 10^6$/0,01	$V_{BL2} = 0{,}28$
$T3$(L_)	$V3 = 43{,}9$	$p3 = 0{,}54$	–	–	–

Объем *Shuffle*, связанный с соединением таблиц, составил: без КИФБ 29,2 ГБ, с КИФБ 5,1 ГБ (эти объемы получены из распечатки работы монитора при выполнении запроса $Q3$). По формуле (9) при $K = 3$ рассчитан прогнозный выигрыш в объеме *Shuffle* с КИФБ: $\Delta = 24{,}5$ ГБ. Фактический выигрыш $\Delta 1 = 29{,}2 - 5{,}1 = 24{,}1$ ГБ. Ошибка прогноза составила $(\Delta - \Delta 1)/\ \Delta 1 \cdot 100 = 2$ %.

Заключение. Проанализирована проблема низкой производительности выполнения запросов в *Spark SQL*, связанная с большим объемом данных, передаваемых по сети при выполнении соединения таблиц базы данных. Известно [11, 12], что лучшими методами решения указанной проблемы являются *SBFCJ* и *SBJ*. Разработана модель, позволившая получить условие, когда тот или иной метод имеет преимущество. Выявлены недостатки метода *SBFCJ*. Показано, что предложенный метод КИФБ можно использовать, в частности, там, где метод *SBFCJ* нельзя применить (*SQL*-запросы к хранилищу данных со схемой типа «снежинка», использование фильтра Блума и дублирование таблиц в одном запросе и др.). Для двух случаев разработаны модели и получены оценки выигрыша в объеме передаваемых по сети данных при использовании метода КИФБ по сравнению с вариантом, когда фильтр Блума не применяется. На конкретном примере проведен анализ адекватности теоретической оценки такого выигрыша.

ЛИТЕРАТУРА

[1] Григорьев Ю.А., Плутенко А.Д., Плужников В.Л. и др. Теория и практика анализа параллельных систем баз данных. Владивосток, Дальнаука, 2015.

[2] Sadalage P., Fowler M. NoSQL Distilled: a brief guide to the emerging world of polyglot persistence. Addison Wesley Professional, 2013.

[3] Perkins L., Redmond E., Wilson J.R. Seven databases in seven weeks: a guide to modern databases and the NoSQL movement. Pragmatic Bookshelf, 2018.

[4] Burdakov A., Grigorev U., Ploutenko A., et al. Estimation models for NoSQL database consistency characteristics. *24th Euromicro Int. Conf. Parallel, Distributed, and Network-Based Processing (PDP)*, 2016, pp. 35–42. DOI: 10.1109/PDP.2016.23

[5] Aslett M. How will the database incumbents respond to NoSQL and NewSQL? *Cs.brown.edu:* веб-сайт.

URL: http://cs.brown.edu/courses/cs227/archives/2012/papers/newsql/aslett-newsql.pdf (дата обращения: 20.03.2019).

[6] Pavlo A., Aslett M. What's really new with NewSQL? *Sigmod Rec.*, 2016, vol. 45, no. 2, pp. 45–55. DOI: 10.1145/3003665.3003674

[7] Dean J., Ghemawat S. MapReduce: simplified data processing on large clusters. *CACM*, 2008, vol. 51, iss. 1, pp. 107–113. DOI: 10.1145/1327452.1327492

[8] White T. Hadoop: The definitive guide. O'Reilly Media, 2015.

[9] Zaharia M., Clowdhury M., Franklin M.J., et al. Spark: cluster computing with working sets. *Proc HotCloud*, 2010, vol. 10, no. 10-10, pp. 1–7.

[10] Karau H., Konwinski A., Wendell P., et al. Learning Spark: lightning-fast big data analysis, O'Reilly Media, 2015.

[11] Karau H., Warren R. High performance Spark: best practices for scaling and optimizing Apache Spark, O'Reilly Media, 2017.

[12] Brito J.J., Mosqueiro T., Ciferri R.R., et al. Faster cloud Star Joins with reduced disk spill and network communication. *Procedia Comput. Sci.*, 2016, vol. 80, pp. 74–85. DOI: https://doi.org/10.1016/j.procs.2016.05.299

[13] Bloom B.H. Space/time trade-offs in hash coding with allowable errors. *CACM*, 1970, vol. 13, iss. 7, pp. 422–426. DOI: 10.1145/362686.362692

[14] Tarkoma S., Rothenberg C.E., Lagerspetz E. Theory and practice of bloom filters for distributed systems. *IEEE Commun. Surv. Tutor.*, 2012, vol. 14, iss. 1, pp. 131–155. DOI: 10.1109/SURV.2011.031611.00024

[15] Клеппман М. Высоконагруженные приложения. Программирование, масштабирование, поддержка. СПб., Питер, 2018.

[16] Григорьев Ю.А., Пролетарская В.А., Ермаков Е.Ю. Метод доступа к хранилищу данных по технологии Spark с каскадным использованием фильтра Блума. *Информатика и системы управления*, 2017, № 1, с. 3–14.

[17] Григорьев Ю.А., Пролетарская В.А., Ермаков Е.Ю. Экспериментальная проверка эффективности метода доступа к хранилищу данных на платформе Spark с каскадным использованием фильтра Блума. *Информатика и системы управления*, 2017, № 3, с. 3–16.

[18] Grigoriev Yu.A., Proletarskaya V.A., Ermakov E.Yu., et al. Efficiency analysis of the access method with the cascading Bloom filter to the data warehouse on the parallel computing platform. *J. Phys.: Conf. Ser.*, 2017, vol. 913, no. 1, art. 012011.
DOI: https://doi.org/10.1088/1742-6596/913/1/012011

[19] [SPARK-21039] [Spark Core] Use treeAggregate instead of aggregate in DataFrame.stat.bloomFilter #18263. *github.com:* веб-сайт.
URL: https://github.com/apache/spark/pull/18263 (дата обращения: 03.04.2019).

[20] RDD.scala. *github.com:* веб-сайт.
URL: https://github.com/apache/spark/blob/master/core/src/main/scala/org/apache/spark/rdd/RDD.scala (дата обращения: 03.04.2019).

[21] Vavilapalli V.K., Murthy A., Douglas C., et al. Apache Hadoop yarn: yet another resource negotiator. *Proc. 4th Ann. Symp. Cloud Computing*. ACM, 2013, art. 5.
DOI: 10.1145/2523616.2523633

[22] TPC-H. *TPC.org:* веб-сайт.
URL: http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.2.pdf (дата обращения: 03.04.2019).

Пролетарская Виктория Андреевна — аспирант кафедры «Системы обработки информации и управления» МГТУ им. Н.Э. Баумана (Российская Федерация, 105005, Москва, 2-я Бауманская ул., д. 5, стр. 1).

Григорьев Юрий Александрович — д-р техн. наук, профессор кафедры «Системы обработки информации и управления» МГТУ им. Н.Э. Баумана (Российская Федерация, 105005, Москва, 2-я Бауманская ул., д. 5, стр. 1).

Просьба ссылаться на эту статью следующим образом:

Пролетарская В.А., Григорьев Ю.А. Модели процессов соединения таблиц хранилища данных по технологии *MapReduce/Spark*. *Вестник МГТУ им. Н.Э. Баумана. Сер. Приборостроение*, 2019, № 5, с. 79–94.
DOI: 10.18698/0236-3933-2019-5-79-94

MODELS OF DATA STORAGE TABLES CONNECTION PROCESSES BY *MapReduce/Spark* TECHNOLOGY

V.A. Proletarskaya victoria.proletarskaya@gmail.com
Yu.A. Grigoryev grigorev@bmstu.ru

Bauman Moscow State Technical University, Moscow, Russian Federation

Abstract

A model has been developed and an estimate of the amount of data transmitted over the network has been obtained with duplicating the table across nodes and using the Bloom filter in the *MapRe-*

Keywords

Big Data, MapReduce, Apache Spark, Spark SQL, Bloom filter, TPC-H, process models, data storage

duce/Spark environment. Models have been developed for fulfilling queries for joining database tables in the cascading Bloom filter in the same environment. Two cases of joining tables are considered: 1) several bushes with one dimension in each of them; 2) one bush with several dimensions — star-type storage. We obtained an estimate of the Bloom filter volume transmitted over the network when the tables are joined. Using the example of the *Q*3 request from the *TPC-H* test, we analyzed the adequacy of the estimated gain in the amount of data transmitted over the network using the cascading Bloom filter. The prediction error was 2 %

Received 28.05.2019

REFERENCES

[1] Grigoryev Yu.A., Plutenko A.D., Pluzhnikov V.L., et al. Teoriya i praktika analiza parallelnykh sistem baz dannykh [Theory and practice of analyzing parallel database systems]. Vladivostok, Dalnauka Publ., 2015.

[2] Sadalage P., Fowler M. NoSQL Distilled: a brief guide to the emerging world of polyglot persistence. Addison Wesley Professional, 2013.

[3] Perkins L., Redmond E., Wilson J.R. Seven databases in seven weeks: a guide to modern databases and the NoSQL movement. Pragmatic Bookshelf, 2018.

[4] Burdakov A., Grigorev U., Ploutenko A., et al. Estimation models for NoSQL database consistency characteristics. *24th Euromicro Int. Conf. Parallel, Distributed, and Network-Based Processing (PDP)*, 2016, pp. 35–42. DOI: 10.1109/PDP.2016.23

[5] Aslett M. How will the database incumbents respond to NoSQL and NewSQL? *Cs.brown.edu:* website. Available at: http://cs.brown.edu/courses/cs227/archives/2012/papers/newsql/aslett-newsql.pdf (accessed: 20.03.2019).

[6] Pavlo A., Aslett M. What's really new with NewSQL? *Sigmod Rec.*, 2016, vol. 45, iss. 2, pp. 45–55. DOI: 10.1145/3003665.3003674

[7] Dean J., Ghemawat S. MapReduce: simplified data processing on large clusters. *CACM*, 2008, vol. 51, iss. 1, pp. 107–113. DOI: 10.1145/1327452.1327492

[8] White T. Hadoop: The definitive guide. O'Reilly Media, 2015.

[9] Zaharia M., Clowdhury M., Franklin M.J., et al. Spark: cluster computing with working sets. *Proc HotCloud*, 2010, vol. 10, no. 10-10, pp. 1–7.

[10] Karau H., Konwinski A., Wendell P., et al. Learning Spark: lightning-fast big data analysis, O'Reilly Media, 2015.

[11] Karau H., Warren R. High performance Spark: best practices for scaling and optimizing Apache Spark, O'Reilly Media, 2017.

[12] Brito J.J., Mosqueiro T., Ciferri R.R., et al. Faster cloud Star Joins with reduced disk spill and network communication. *Procedia Comput. Sci.*, 2016, vol. 80, pp. 74–85. DOI: https://doi.org/10.1016/j.procs.2016.05.299

[13] Bloom B.H. Space/time trade-offs in hash coding with allowable errors. *CACM*, 1970, vol. 13, iss. 7, pp. 422–426. DOI: 10.1145/362686.362692

[14] Tarkoma S., Rothenberg C.E., Lagerspetz E. Theory and practice of bloom filters for distributed systems. *IEEE Commun. Surv. Tutor*., 2012, vol. 14, iss. 1, pp. 131–155. DOI: 10.1109/SURV.2011.031611.00024

[15] Kleppmann M. The big ideas behind reliable, scalable, and mantainable systems. O'Reilly Media, 2017.

[16] Grigoryev Yu.A., Proletarskaya V.A., Ermakov E.Yu. Access method to the warehouse using Spark technology with cascading Bloom filter. *Informatika i sistemy upravleniya*, 2017, no. 1, pp. 3–14 (in Russ.).

[17] Grigoryev Yu.A., Proletarskaya V.A., Ermakov E.Yu. Experimental efficiency verification of an access method to the storage data on the Spark platform using cascading Bloom filter. *Informatika i sistemy upravleniya*, 2017, no. 3, pp. 3–16 (in Russ.).

[18] Grigoriev Yu.A., Proletarskaya V.A., Ermakov E.Yu., et al. Efficiency analysis of the access method with the cascading Bloom filter to the data warehouse on the parallel computing platform. *J. Phys.: Conf. Ser*., 2017, vol. 913, no. 1, art. 012011. DOI: https://doi.org/10.1088/1742-6596/913/1/012011

[19] [SPARK-21039] [Spark Core] Use treeAggregate instead of aggregate in DataFrame.stat.bloomFilter #18263. *github.com:* website.
Available at: https://github.com/apache/spark/pull/18263 (accessed: 03.04.2019).

[20] RDD.scala. *github.com:* website. Available at: https://github.com/apache/spark/blob/master/core/src/main/scala/org/apache/spark/rdd/RDD.scala (accessed: 03.04.2019).

[21] Vavilapalli V.K., Murthy A., Douglas C., et al. Apache Hadoop yarn: yet another resource negotiator. *Proc. 4th Ann. Symp. Cloud Computing*. ACM, 2013, art. 5. DOI: 10.1145/2523616.2523633

[22] TPC-H. *TPC.org:* website.
Available at: http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.2.pdf (accessed: 03.04.2019).

Proletarskaya V.A. — Post-Graduate Student, Department of Information Processing Systems, Bauman Moscow State Technical University (2-ya Baumanskaya ul. 5, str. 1, Moscow, 105005 Russian Federation).

Grigoryev Yu.A. — Dr. Sc. (Eng.), Professor, Department of Information Processing Systems, Bauman Moscow State Technical University (2-ya Baumanskaya ul. 5, str. 1, Moscow, 105005 Russian Federation).

Please cite this article in English as:

Proletarskaya V.A., Grigoryev Yu.A. Models of data storage tables connection processes by *MapReduce/Spark* technology. *Herald of the Bauman Moscow State Technical University, Series Instrument Engineering*, 2019, no. 5, pp. 79–94 (in Russ.). DOI: 10.18698/0236-3933-2019-5-79-94