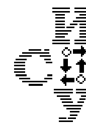

УДК 004.657

© 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»)

ЭКСПЕРИМЕНТАЛЬНАЯ ПРОВЕРКА ЭФФЕКТИВНОСТИ МЕТОДА ДОСТУПА К ХРАНИЛИЩУ ДАННЫХ НА ПЛАТФОРМЕ SPARK С КАСКАДНЫМ ИСПОЛЬЗОВАНИЕМ ФИЛЬТРА БЛУМА

На примере SQL-запроса Q3 теста TPC-H выполнено сравнение двух методов доступа к хранилищу данных: разработанного метода с каскадным использованием фильтра Блума (КИФБ) и без использования фильтра Блума (БИФБ). С этой целью проведены натурные эксперименты в среде кластера из 8 узлов на платформе параллельных вычислений Apache Spark. Результаты экспериментов подтвердили преимущества разработанного метода КИФБ.
Ключевые слова: Spark SQL, фильтр Блума, TPC-H, запрос Q3, сравнение методов.

DOI: 10.22250/isu.2017.53.3-16

Введение

Реляционные базы данных (РБД) до сих пор сохраняют лидирующие позиции во многих областях обработки данных. Но с появлением технологий обработки больших объемов данных (Big Data) стали проявляться недостатки РБД: ограниченная масштабируемость, невозможность хранения агрегатов в явном виде и др. С целью преодолеть ограничения РБД появился новый класс систем, который принято называть NoSQL [1].

Базы данных NoSQL обладают высокой горизонтальной масштабируемостью (тысячи узлов) и устраняют некоторые ограничения реляционной модели (позволяют хранить агрегаты в явном виде, нет единой схемы базы данных и др.), но они не поддерживают ведение транзакций, что важно для финансовых систем.

Чтобы преодолеть этот недостаток NoSQL, были созданы базы данных NewSQL [2]. Они поддерживают реляционную модель, включая ведение транзакций, и одновременно обладают высокой горизонтальной масштабируемостью. Но при этом NewSQL наследуют недостатки реляционной модели: здесь нельзя хранить агрегаты в явном виде, необходимо разрабатывать единую схему базы данных.

Вслед за NoSQL появились платформы, поддерживающих технологию MapReduce (MR) [3]. Они обеспечивают параллельную обработку запросов (в частности SQL-запросов) на кластерах большой размерности (Hadoop – 4000 узлов). Их преимуществом является также большое число реплик и, следовательно, высокая надежность обработки огромных объемов данных.

В настоящее время ведутся интенсивные работы в направлении расширения возможностей технологии MapReduce. Так, наряду с Hadoop, большую известность получила новая программная платформа Spark [4]. Как и Hadoop, Spark – это не просто отдельный проект, а целая экосистема основанных на нем разнообразных проектов, например:

Spark SQL, предоставляющий SQL-подобную функциональность на базе Spark, проект интегрирован с Hive;

Spark Streaming для обработки потоков;

Spark GraphX для осуществления вычислений на графах;

Spark MLib для машинного обучения.

Сравним два рассмотренных подхода, Hadoop и Spark, по производительности, устойчивости к сбоям и простоте программирования [5].

1. Запросы в Spark выполняются быстрее. В Hadoop промежуточные данные сохраняются и на локальном диске (перед shuffle), и в файловой системе HDFS. В Spark промежуточные данные сохраняются на локальном диске только перед shuffle («перетасовкой»). Результаты выполнения join сохраняются в ОП (при достаточном объеме оперативной памяти). Конечно, при этом расход ОП намного выше.

2. Устойчивость к сбоям в Hadoop выше. Если произойдет сбой узла, то Hadoop перезапустит функцию (map или reduce), которая выполняет только часть работы, на другом узле. В Spark весь запрос выполняется с начала, если все промежуточные данные хранятся в ОП.

3. В Spark проще программировать сложные процессы обработки (отличные от простых запросов Select). В Spark достаточно последовательно закодировать требуемые операции map, join, save и реализовать функции map на языке высокого уровня. В Hadoop необходим кодировать функции map и reduce для каждого задания на языке Java.

Система Spark отлично масштабируется, обработка на каждом этапе ведется параллельно на многих серверах. Это позволяет обеспечить высокую эффектив-

ность обработки входного потока данных и запросов операторов.

В [6] предлагается решение, которое позволяет существенно уменьшить время выполнения сложных аналитических запросов к большому хранилищу данных. Оно основано на применении фильтра Блума [3] для хранилищ со схемой «звезда».

Авторами был разработан метод параллельной обработки запроса к хранилищу данных с произвольной структурой (не обязательно «звезда») с каскадным использованием фильтра Блума (КИФБ) [7]. В статье приведены результаты экспериментальной проверки эффективности разработанного метода на примере запроса Q3 из теста TPC-H [8].

Применение фильтра Блума для хранилища данных со схемой «звезда»

Фильтр Блума – это специально построенный массив битов на основе хеширования значений ключа таблицы измерения. Внешний ключ каждой записи таблицы фактов обрабатывается фильтром Блума [7]. Этот фильтр дает «осечку» с вероятностью $1/2^K$, где K – параметр фильтра Блума. При достаточно большом значении K можно добиться, чтобы вероятность ложноположительного срабатывания фильтра была бы очень небольшой. В этом случае уменьшается число записей, участвующих в соединении, а в некоторых случаях вообще не требуется выполнять соединение таблицы измерений с таблицей фактов. Это позволяет существенно уменьшить время обработки запросов.

Фильтр Блума уже применяется на практике. Brito и другие показали эффективность использования фильтров Блума на примере хранилища данных типа «звезда» [6]. Этот метод (SBFCJ – Spark Bloom-Filtered Cascade Join) выигрывает по времени реакции (Elapsed time) и объему данных, пересылаемых между узлами (shuffled bytes) и дополнительно сохраняемых на диске (disk spill), по сравнению с методами, реализованными в MapReduce (MR) (рис.1, прямоугольник).

Конкуренцию ему составляет метод SBJ (Spark Broadcast Join), где все таблицы измерений целиком кэшируются в оперативной памяти на каждом узле (рис. 1, кружок).

Но при использовании метода SBJ наблюдаются сильные флуктуации объемов используемой оперативной памяти (ОП) при достаточно больших размерах хранилища и небольших объемах ОП [6].

На рис. 2 приведены зависимости времени выполнения запроса Q4.1 из теста Star Schema Benchmark (SSB) [9] от объема ОП, предоставляемого одному executor (Java-машине). Эксперимент выполнялся с 20 executor.

База данных теста SSB имеет схему типа «звезда» (4 измерения, 1 таблица фактов); запрос Q4.1 – это соединение 4-х измерений и таблицы фактов, группи-

рование, сортировка; запрос выполнялся для наполнения SF = 200: 1.2 миллиарда записей в таблице фактов (т.ф.), объем т.ф.– 130 Гб. Кластер имел следующие характеристики: количество узлов – 21 (1 master и 20 slaves), Hadoop 2.6.0 и Apache Spark 1.4.1, ОС GNU/Linux installation (CentOS 5.7), каждый узел – 2GHz AMD CPUs и 8GB ОП.

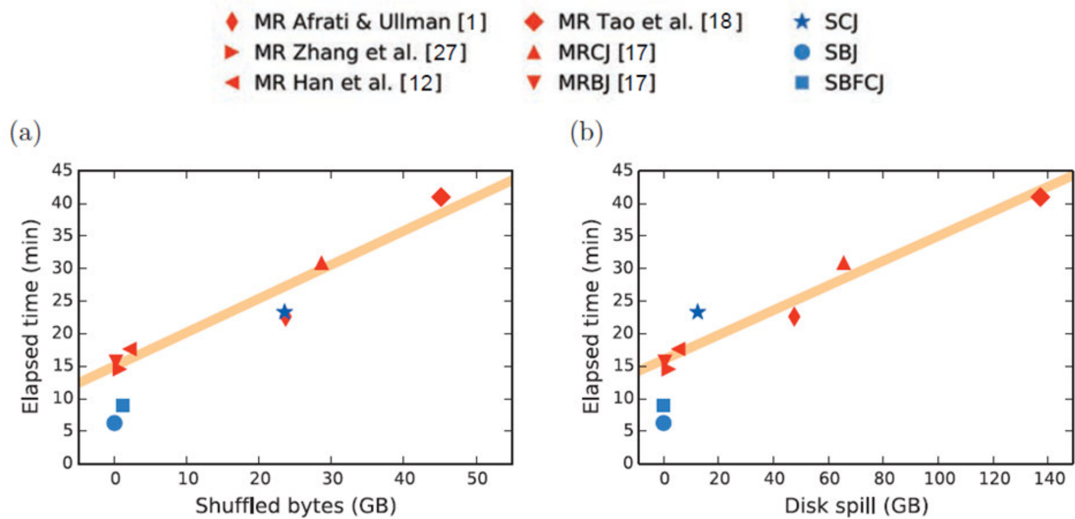

Рис. 1. Время реакции системы в зависимости от объема данных, перемещаемых по сети (a), и объема данных, сохраняемых на диске из-за нехватки ОП (b) [6].

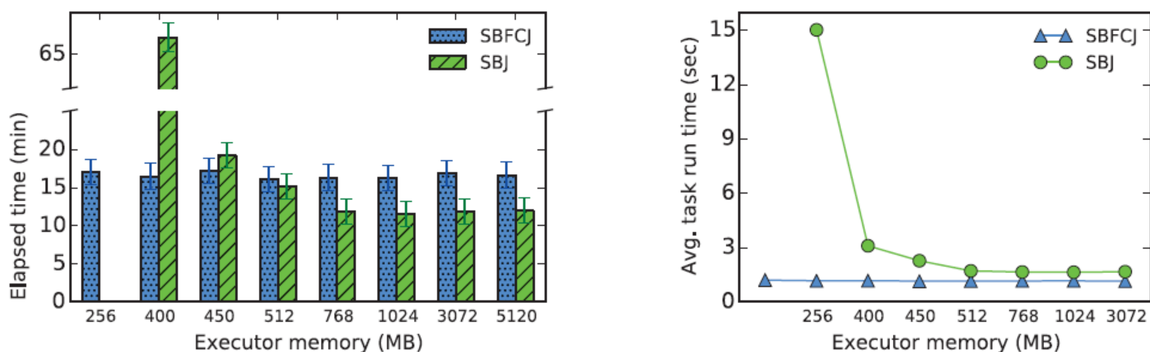

Рис. 2. Сравнение характеристик SBJ и SBFCJ с 20 исполнителями (executor) в зависимости от объема ОП на один executor [6]: времени выполнения запроса и среднего времени на задачу.

При объеме ОП на один executor меньше 400 Мб наблюдается резкое увеличение времени выполнения запроса и отдельной задачи при использовании метода SBJ. Таким образом, метод SBFCJ более устойчив при изменении объема ОП. Поэтому фильтр Блума был выбран авторами для разработки метода параллельной обработки запроса к хранилищу данных с произвольной структурой.

Описание метода доступа к хранилищу данных с каскадным использованием фильтра Блума (КИФБ)

Доступ к хранилищу данных с использованием метода КИФБ проиллюстрируем на примере запроса Q3 из теста TPC-H [8] (рис. 3).

```
select l_orderkey, sum(l_extendedprice*(1-l_discount)) as revenue,
o_orderdate, o_shippriority
from customer, orders, lineitem
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;
```

Рис. 3. Запрос Q3 из теста TPC-H.

Тест TPC-H оценивает производительность систем поддержки принятия решений (СППР). Он состоит из набора сложных, бизнес-ориентированных запросов ad hoc. Данные в таблицах и запросы подобраны так, чтобы отражать некоторую усредненную по индустрии бизнес-активность. Типичные запросы составлены так, чтобы соответствовать основным типам запросов в СППР: ценообразование и скидки, управление прибылью, исследование предпочтений покупателей, исследование рынка и т.п. В частности, Q3 представляет собой «запрос приоритета доставки». Он извлекает приоритет доставки и потенциальный доход, определяемый как сумма l_extendedprice * (1-l_discount), из заказов, имеющих наибольший доход среди тех, которые не были отправлены на определенную дату. Заказы перечисляются в порядке убывания дохода. Если существует более 10 отгруженных заказов, выводятся только 10 заказов с наибольшим доходом. В табл. 1 приведены характеристики таблиц, участвующих в запросе Q3, SF – коэффициент наполнения [8].

Таблица 1

Имя таблицы	Число записей	Средняя длина записи (байты)	Средний размер таблицы (Мбайт)
CUSTOMER	SF*150 000	179	SF*26
ORDERS	SF*1 500 000	104	SF*149
LINEITEM	SF*6 001 215	112	SF*641

На рис. 4 приведен граф соединения и преобразования подзапросов исходного запроса Q3 [7], где прямоугольники обозначают исходные таблицы; прямоугольники с закругленными углами – RDD-таблицы, которые формируются Spark после выполнения подзапросов C1, O1, L1; прямоугольники с пунктирной границей – результаты соединения RDD-таблиц подзапросов; овал – дополнительные операции (группирование, упорядочивание); в круглых скобках указаны используемые атрибуты исходных таблиц. Дуги со стрелками обозначают исходные данные для формирования RDD-таблиц или выполнения операций, дуги без стрелок – условия соединений соответствующих RDD-таблиц. Кружками обозначены фильтры Блума (над ними указаны атрибуты, для которых строятся эти фильтры).

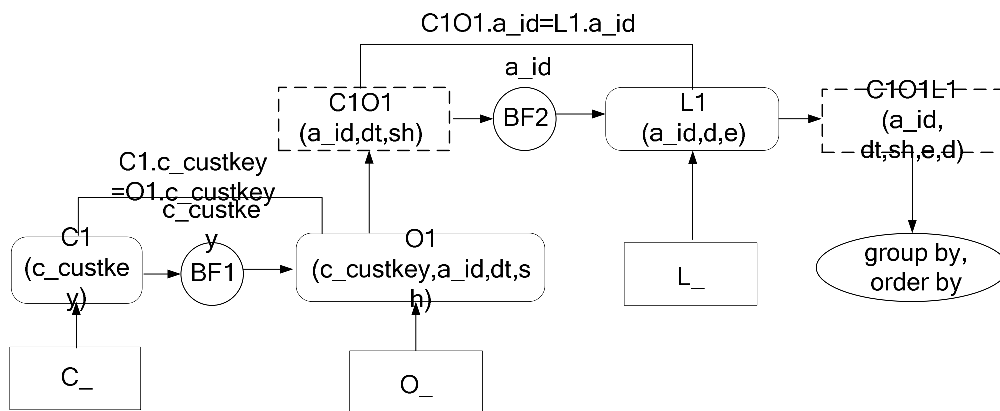

Рис. 4. Граф соединения и преобразования подзапросов для Q3 с использованием метода КИФБ.

На рис. 4 приняты следующие обозначения: C_ , O_, L_ – исходные таблицы customer, orders, lineitem; C1, O1, L1 – подзапросы Select, применяемые к таблицам C_, O_ и L_ (см. пояснения к программе, представленной на рис. 5); a_id – атрибут o_orderkey в таблице O_ и атрибут l_orderkey в таблице L_; dt, sh, d, e – соответственно атрибуты o_orderdate, o_shippriority, l_discount, l_extendedprice.

Из рис. 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»: C1-O1 (измерение C1, таблица фактов O1) и C1O1-L1 (измерение C1O1, таблица фактов L1). Из описания последовательности выполнения исходного запроса Q3 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. соединение реализуется с уменьшенными по объему таблицами фактов.

На рис. 5 представлена программа на языке Scala (реализующая граф, представленный на рис. 4), где 1 – подключение внешних библиотек; 2 – установка числа задач reduce, используемых при соединении таблиц, равным 14 (7 рабочих узлов × 2 ядра); 3 – подключение базы данных TPC-H с требуемым коэффициентом наполнения SF (на рис. 5 SF =100); 4 – сохранение времени начала выполнения запроса Q3; 5 – выполнение подзапроса C1 (метод persist позволяет в дальнейшем ссылаться на построенный набор RDD customer без повторного его создания); 6 – построение фильтра Блума BF1 (переменная bloom1) и создание широковещательной переменной broadbloom1 для автоматической рассылки фильтра Блума BF1 исполнительным процессам (executor); 7 – выполнение подзапроса O1 и фильтрация записей с помощью фильтра Блума BF1; 8 – соединение таблиц подзапросов C1 и O1 после фильтрации, чтобы исключить записи, связанные с ложноположительным срабатыванием фильтра Блума [7]; 9 – построение фильтра Блума BF2 (переменная bloom2) и создание широковещательной переменной

broadbloom2 для рассылки фильтра Блума; 10 – выполнение подзапроса L1 и фильтрация записей с помощью фильтра Блума BF2; 11 – соединение таблиц C1O1 и L1 по атрибуту a_id после фильтрации (см. рис. 4); 12 – реализация операций group by и order by; 13 – сохранение времени окончания выполнения запроса Q3.

1	import java.util.Calendar import breeze.util.BloomFilter import org.apache.spark.storage.StorageLevel import org.apache.spark.sql.functions
2	val sqlContext = new org.apache.spark.sql.SQLContext(sc) sqlContext.setConf("spark.sql.shuffle.partitions", "14")
3	val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc) hiveContext.setConf("spark.sql.orc.filterPushdown", "true") hiveContext.sql("use tpch_flat_orc_100")
4	val time = Calendar.getInstance().getTime()
5	val customer = hiveContext.sql("SELECT c_custkey FROM customer where c_mktsegment = 'BUILDING'").persist(StorageLevel.MEMORY_AND_DISK)
6	val bloom1 = customer.stat.bloomFilter($"c_custkey", 15000000, 0.01) val broadbloom1 = sc.broadcast(bloom1)
7	val orders = hiveContext.sql("SELECT o_custkey as c_custkey, o_orderkey as a_id, o_orderdate, o_shippriority FROM orders where o_orderdate< '1995-03-15'").filter(line=>broadbloom1.value.mightContain(line(0)))
8	val join1 = customer.join(orders,Seq("c_custkey")).persist(StorageLevel.MEMORY_AND_DISK)
9	val bloom2 = join1.stat.bloomFilter($"a_id", **15000000**, 0.01) val broadbloom2 = sc.broadcast(bloom2)
10	val lineitem = hiveContext.sql("SELECT l_orderkey as a_id, l_extendedprice, l_discount FROM lineitem where l_shipdate> '1995-03-15'").filter(line=>broadbloom2.value.mightContain(line(0)))
11	val join2 = lineitem.join(join1,Seq("a_id"))
12	val join2_r = join2.withColumn("revenue", (- join2("l_discount")+1)*join2("l_extendedprice")) val aggr = join2_r.groupBy("a_id","o_orderdate","o_shippriority").agg(sum("revenue")) aggr.orderBy(desc("sum(revenue)"),asc("o_orderdate")).show(10) aggr.persist(StorageLevel.MEMORY_AND_DISK)
13	val time2 = Calendar.getInstance().getTime()

Рис. 5. Текст программы драйвера, реализующей запрос Q3 с использованием метода КИФБ.

Описание метода реализации запросов в Spark SQL без использования фильтра Блума (БИФБ)

Разработанный метода доступ к хранилищу данных с каскадным использованием фильтра Блума (КИФБ) сравнивался с методом реализации запросов в

Spark SQL без использования фильтра Блума (БИФБ). Доступ к хранилищу данных по методу БИФБ проиллюстрируем на примере запроса Q3 из теста TPC-H (рис. 3). На рис. 6 представлена программа на языке Scala, реализующая запрос Q3, где 1 – подключение внешних библиотек; 2 – установка числа задач reduce, используемых при соединении таблиц, равным 14; 3 – подключение базы данных TPC-H с требуемым коэффициентом наполнения SF (на рис. 6 SF = 100): 4 – сохранение времени начала выполнения запроса Q3; 5 – выполнение исходного запроса Q3, представленного на рис. 3 (исходный запрос Select целиком кодируется как строка при вызове метода hiveContext.sql); 6 – сохранение времени окончания выполнения запроса Q3.

1	import java.util.Calendar import breeze.util.BloomFilter import org.apache.spark.storage.StorageLevel import org.apache.spark.sql.functions
2	val sqlContext = new org.apache.spark.sql.SQLContext(sc) sqlContext.setConf("spark.sql.shuffle.partitions", "14")
3	val hiveContext = new org.apache.spark.sql.hive.HiveContext(sc) hiveContext.setConf("spark.sql.orc.filterPushdown", "true") hiveContext.sql("use tpch_flat_orc_100")
4	val time = Calendar.getInstance().getTime()
5	val customer = hiveContext.sql("SELECT l.l_orderkey,sum(l.l_extendedprice*(1-l.l_discount)) as revenue,o.o_orderdate,o.o_shippriority FROM customer c, orders o, lineitem l where o.o_custkey=c.c_custkey and l.l_orderkey=o.o_orderkey and c_mktsegment = 'BUILDING' and o_orderdate< '1995-03-15' and l_shipdate> '1995-03-15' group by l.l_orderkey,o.o_orderdate,o.o_shippriority order by revenue desc, o.o_orderdate").persist(StorageLevel.MEMORY_AND_DISK) customer.show(10)
6	val time2 = Calendar.getInstance().getTime()

Рис. 6. Текст программы драйвера, реализующей запрос Q3 в Spark SQL без использования фильтра Блума.

Описание экспериментальной установки и анализ результатов натурных экспериментов

Для проведения экспериментального сравнения рассматриваемых методов БИФБ и КИФБ был развернут виртуальный кластер. Кластер состоял из 8 узлов, на одном из которых были инсталлированы управляющие службы HDFS, Hive, Spark, Yarn. Основные характеристики каждого виртуального узла были следующими: один двуядерный процессор, 8 GB оперативной памяти, 200 GB SSD диск, OC Ubuntu 14.16.

Рассмотрим сначала результаты выполнения запроса Q3 (см. рис. 3) с использованием метода БИФБ. На рис. 7 приведены стадии (stage) БИФБ и время их реализации в Spark для SF=500 (это соответствует примерно 400 Гб обрабатываемых данных). Знаком ►◄ обозначена операция соединения таблиц.

	Стадии БИФБ	**Время (мин.) выполнения, SF=500**
0	Чтение L_ (Stage0)	9,3
1	Чтение C_(Stage1)	1,0
2	Чтение O_(Stage2)	9,2
3	C_►◄O_(Stage3)	1,1
4	(C_►◄O_)►◄L_ (Stage4)	7,9
9	group by (Stage9)	6,7
10	order by(Stage10)	0,02
		ИТОГО: 9,3+1,1+7,9+6,7+0,02=25 мин

Рис. 7. Стадии БИФБ.

Можно заметить, что чтение исходных таблиц L_, C_, O_ выполняется параллельно. Время выполнения запроса составило 25 мин.

В табл. 2 приведены характеристики выполнения задач (task) БИФБ по стадиям выполнения (SF = 500). Исполнитель (executor) – это виртуальная машина Java (JVM), на которой выполняются задачи стадии. По умолчанию число ядер CPU на один экземпляр исполнителя равно 1 (параметр spark.executor.cores).

В табл. 2 стрелками показано, что чтение исходных таблиц (стадии 0÷2) выполняется параллельно на 13 исполнителях (13 ядер CPU на 7 узлах).

Среднее число задач, выполняемых одним ядром, в этом случае равно (455+24+100)/13 = 44,5. Из анализа квантилей времени выполнения одной задачи следует, что задачи стадий 0÷2 имеют длинный правый хвост функции распределения, а задачи стадий 4,9,10 – длинный левый хвост. Можно также заметить, что «сбор мусора» и запись на диск перед «перетасовкой» (Shuffle Write) не оказывают существенного влияния на время выполнения запроса.

Таблица 2

SF=500	**Число**			**Квантили времени выполнения одной задачи, сек.**					**Среднее время на задачу, сек.**	
Стадии БИФБ	**задач**	**executor**	**задач/ядро**	**min**	**0,25**	**0,5**	**0,75**	**max**	**сбор мусора**	**Shuffle Write**
0	455	13		0,2	8	12	16	49	0,028	0,1
1	24	2	44,5	0,027	2	3	6	10	0,011	0,055
2	100	13		3	7	10	15	28	0,037	0,2
3	14	13	1,1	31	36	40	42	45	1	0,5
4	14	13	1,1	210	300	318	324	342	4	
9	14	13	1,1	144	264	276	282	288	6	0,028
10	1	1	1	1	1	1	1	1	0,026	

В табл. 3 приведены объемы данных и число записей, прочитанных из файловой системы HDFS (Input), считанных с локальных дисков и переданных по виртуальной сети на этапе «перетасовки» при выполнении задач reduce (Shuffle Read), записанных на локальные диски перед «перетасовкой» (Shuffle Write) всеми задачами соответствующих стадий. Стрелками обозначены составляющие соответствующих объемов Shuffle Read.

Следует отметить высокий коэффициент сжатия исходных таблиц. В среднем он составил (см. табл. 1 и 3) 500*(26+149+641)/(32,1*1024) =12,4.

Таблица 3

Стадии БИФБ	SF=500: Объем(Гб)/Число записей			
	Input	Shuffle Read	Shuffle Write	
0	26,2/3 000 028 242		23,7/1 617 261 771	
1	0,177/75 000 000		0,070/14 997 607	
2	5,70/750 000 000		4,40/364 396 568	
3		4,50/379 394 175	1,00/72 858 348	
4		24,7/1 690 120 119		
9		24,7/1 690 120 119	0,109/5 658 551	
10		0,006/337 532		
ИТОГО:	**32,1/3 825 028 242**	**53,9/3 759 971 945**	**29,3/2 075 172 845**	

Теперь рассмотрим результаты выполнения того же запроса Q3 (см. рис. 3), но с использованием разработанного метода КИФБ. На рис. 8 приведены стадии (stage) КИФБ и время их реализации в Spark для SF = 500, числа элементов в фильтре Блума N = 40 млн и вероятности ложноположительного срабатывания фильтра P = 0,01 [7]. На Stage2 запускаются задачи map, которые сканируют набор RDD "customer" (in MemoryTableScan RDD), уже созданный на стадии Stage0. При этом читаются некоторые данные из HDFS (0,079Гб/1501, см. табл. 5 – это не таблица C_). Эти задачи помещают записи RDD "customer" на локальный диск (Shuffle Write) для дальнейшей «перетасовки» и соединения с таблицей O_ на стадии Stage3. Задачи стадий Stage1 и Stage2 выполняются параллельно. Аналогичные действия Spark выполняет на стадии Stage6 с таблицей C1O1. Задачи стадий Stage6 и Stage7 выполняются параллельно. Время выполнения запроса составило 11 мин., что в 25/11 = 2,3 раза меньше, чем без использования фильтра Блума (см. рис. 7).

В табл. 4 приведены характеристики выполнения задач (task) КИФБ по стадиям выполнения (SF = 500, N = 40 млн, P = 0,01). Стрелкой показано, что стадии 1 и 2 выполняются параллельно на 13 исполнителях. Среднее число задач, выполняемых одним ядром, в этом случае равно (100 + 24)/13 = 9,5. Стадии 6 и 7 также выполняются параллельно.

	Стадии КИФБ	Время (мин.) выполнения, SF=500, N=40 млн.
0	Чтение C_ и построение фильтра Блума bloom1 (Stage0)	0,4
1	Чтение O_ и применение фильра bloom1 (Stage1)	1,5
2	map C_ – для соединения с O_ (Stage2)	0,3
3	C_►◄O_ и построение фильтра Блума bloom2 (Stage3)	0,6
6	map C_►◄O_ – для соединения с L_ (Stage6)	0,1
7	Чтение L_ и применение фильтра bloom2 (Stage7)	7,3
8	L_►◄ (C_►◄O_), group by, order by (Stage8)	1,2
		ИТОГО: 0,4+1,5+0,6+7,3+1,2=11 мин

Рис. 8. Стадии КИФБ.

Таблица 4

SF=500 N=40	Число			Квантили времени выполнения одной задачи, сек.					Среднее время на задачу, сек.	
Стадии КИФБ	задач	executor	задач/ядро	min	0,25	0,5	0,75	max	сбор мусора	Shuffle Write
0	24	13	1,8	0,8	2	4	6	9	0,3	
1	100	13	9,5	3	4	9	12	19	0,1	0,063
2	24	2		0,004	0,029	0,041	0,1	3	0	0,3
3	14	13	1,1	16	19	19	19	20	0,8	
6	14	13	36,1	2	2	2	2	3	0,2	0,2
7	455	13		0,1	7	10	13	53	0,1	0,03
8	14	13	1,1	21	53	55	59	60	0,8	

В табл. 5 приведены объемы данных и число записей для Input, Shuffle Read и Shuffle Write. Следует отметить, что объемы Shuffle Read и Shuffle Write при использовании метода КИФБ меньше по сравнению с БИФБ соответственно в 10,5 и 5,7 раза (см. табл. 3).

Таблица 5

Стадии КИФБ	SF=500, N=40 млн.: Объем(Гб)/Число записей		
	Input	Shuffle Read	Shuffle Write
0	0,177/75 000 000		
1	5,70/750 000 000		0,944/75 783 663
2	0,079/1 501		0,070/14 997 607
3		1,014/90 781 270	
6	0,847/7 291		1,006/72 858 348
7	26,2/3 000 028 242		3,1/203 379 624
8		4,1/276 237 972	
ИТОГО:	**33,0/3 825 037 034**	**5,11/367 019 242**	**5,11/367 019 242**

На рис. 9 показано влияние параметра N (млн) фильтра Блума на показатели выполнения запроса Q3 с КИФБ.

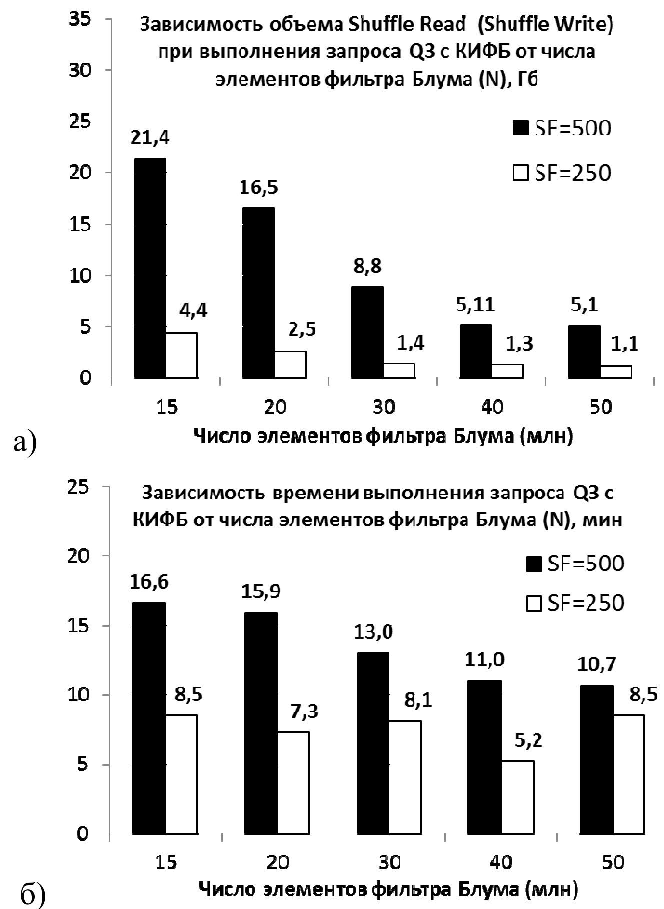

Рис. 9. Зависимость объема Shuffle Read (Shuffle Write) (а)и времени выполнения запроса (б) от числа элементов фильтра Блума N (млн).

Объем Shuffle Read (Shuffle Write) существенно уменьшается при увеличении N до 30 млн для SF = 250 и до 40 млн – для SF = 500. Дальнейший рост N не приводит к существенному уменьшению объема. Это связано с сокращением числа ложноположительных срабатываний фильтра Блума. Время, показанное на рис. 9 б, является случайным (сказывается, что кластер развернут в виртуальной среде). Но при SF = 500 наблюдается тенденция к уменьшению времени выполнения запроса до N = 40 млн. В данном эксперименте влияние N на время выполнения запроса ниже, чем на объем данных «перетасовки»: для SF = 500 при увеличении N с 15 млн до 40 млн объем уменьшился в 21,4/5,11 = 4,2 раза, а время – только в 16,6/11 = 1,5 раза. Это связано с тем, что сеть между узлами является виртуальной, т.е. «быстрой» (некоторые узлы располагаются на одном физическом сервере).

Из рис. 10 видно, что объем промежуточных данных, сохраняемых на локальном диске и передаваемых по сети в процессе «перетасовки» (Shuffle Read), существенно выше для метода БИФБ, чем для КИФБ.

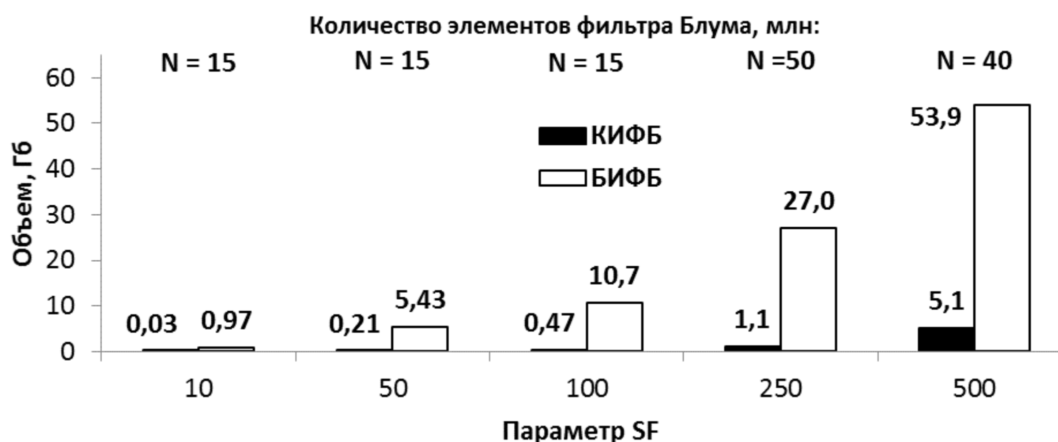

Рис. 10. Зависимость объема Shuffle Read от SF.

Метод БИФБ проигрывает КИФБ и по времени выполнения запроса (рис. 11). До SF = 250 БИФБ незначительно уступает КИФБ. Но при SF = 500 выигрыш КИФБ очевиден.

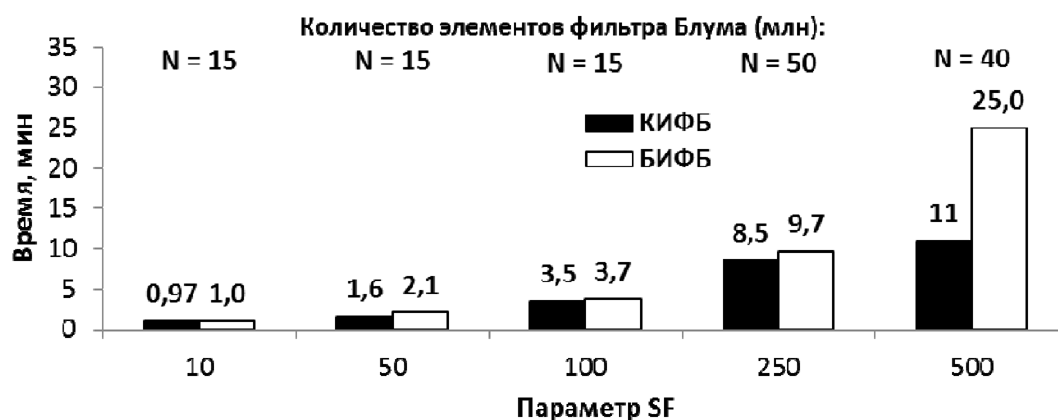

Рис. 11. Зависимость времени выполнения запроса от SF.

Заключение

Разработаны программы драйвера на языке Scala, реализующие SQL-запрос Q3 из теста TPC-H в среде Spark: с каскадным использованием фильтра Блума (КИФБ, рис. 5) и без использования фильтра Блума (БИФБ, рис. 6).

Проведена серия натурных экспериментов на кластере из 7 рабочих узлов, реализованном в облачной архитектуре, с целью сравнения методов КИФБ и БИФБ доступа к хранилищу данных на примере запроса Q3.

При использовании метода КИФБ объем промежуточных данных, сохраняемых на диске и передаваемых по сети при выполнении запроса, намного меньше. Так, для SF = 500 эти объемы отличаются на порядок (см. рис. 10), т.е. при выполнении запроса нагрузка на диск и сеть для метода КИФБ меньше.

КИФБ выигрывает и по времени выполнения запроса. Для SF = 500 метод КИФБ лучше БИФБ по времени в 2,3 раза (см. рис. 11).

Ожидается, что эффект будет еще выше, если кластер реализовать не в виртуальной среде. Показано, что параметр N фильтра Блума существенно влияет на показатели выполнения запроса с использование метода КИФБ (см. рис. 9).

Далее предполагается разработать общую стоимостную модель для метода доступа КИФБ и использовать приведенные в этой статье результаты измерений для адаптации модели и оценки ее адекватности.

ЛИТЕРАТУРА

1. *Венкат Гудивада, Дана Рао, Виджай Рагхаван*. Ренессанс СУБД: проблема выбора // Открытые системы. СУБД. – 2016. – № 3. – С.12-17.
2. *Нелюбин Д., Косьяненко Г.* ЧТО НОВОГО В NEWSQL? Погружение в новейшие базы данных // Хакер. – 2014. – № 8 (187). – С.127-131.
3. *Григорьев Ю.А., Плутенко А.Д., Плужников В.Л., Ермаков Е.Ю., Цвященко Е.В., Пролетарская В.А.* Теория и практика анализа параллельных систем баз данных. – Владивосток: Дальнаука, 2015.
4. *Григорьев Ю.А.* Технологии аналитической обработки больших данных // Информационно-измерительные и управляющие системы. – 2016. – Т.14, № 12. – С.59-68.
5. *J. Shi, Y. Qiu, U. F. Minhas, L. Jiao, C. Wang, B. Reinwald, and F.* ¨Ozcan, “Clash of the titans: Mapreduce vs. spark for large scale data analytics,” Proceedings of the VLDB Endowment. – 2015. – Vol. 8, No. 13. – P.2110-2121,.
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., Ермаков Е.Ю.* Метод доступа к хранилищу данных по технологии SPARK с каскадным использованием фильтра Блума // Информатика и системы управления. – 2017. – № 1(51). – С.3-14.
8. Документация TPC-H // http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.17.2.pdf.
9. *O’Neil P. et al.* The star schema benchmark and augmented fact table indexing //Technology Conference on Performance Evaluation and Benchmarking. – Springer Berlin Heidelberg, 2009. – С.237-252.

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