Повышение эффективности Clusterix-подобных СУБД для аналитической обработки больших данных

Р. К. Классен, В. А. Райхлин

Казанский национальный исследовательский технический университет им. А.Н. Туполева – КАИ, г. Казань, Россия

Аннотация. Коммерческие OLAP-системы экономически недоступны организациям с ограниченными финансовыми возможностями. Аналитическую обработку данных значительных объемов в этих организациях можно осуществить с использованием open source программных систем на экономичной кластерной платформе. Ранее созданные Clusterix-подобные СУБД были недостаточно эффективны по критерию «производительность/стоимость». С целью повышения эффективности подобных систем в статье рассматривается их дальнейшее развитие путем полной загрузки процессорных ядер в комплексе с GPU-акселерацией (системы Clusterix-N, N – от New) вплоть до разработки системы, сравнимой по эффективности с открытой системой Spark, полагаемой в настоящее время наиболее перспективной. За основу развития была принята методология конструктивного моделирования систем.

Ключевые слова: аналитическая обработка данных значительных объемов, open source программные системы на кластерной платформе, повышение эффективности Clusterix-подобных СУБД, полная загрузка процессорных ядер, GPU-акселерация, сравнение со Spark, принятая методология.

DOI 10.14357/20718632190405

Введение

Объемы баз данных в сотни GB и более нередки для относительно небольших предприятий с ограниченными финансовыми возможностями. Приобретение такими организациями экономичных вычислительных кластеров и специализированного ПО СУБД консервативного типа (с эпизодическим обновлением данных) делает возможной для них своевременную обработку накопленных данных. Для консервативных СУБД свойственна OLAP нагрузка [1], характеризующаяся высоким удельным весом сложных запросов типа «селекция – проекция – соединение», оперирующих множеством таблиц с большим числом операций соединения. Разработки в этом направлении ведутся. Коммерческие СУБД обладают высокой производительностью и надежностью, но чрезмерно большой стоимостью. Так, СУБД MS SQL Server 2016 [2, 3] на одном сервере Lenovo x3950X6 [4] имеет совокупную стоимость системы $2 634 342 (сервер ~$1,5 млн. + ПО ~$1 млн.). СУБД Oracle Database [5] с расширением для OLAP и лицензией на 384 ядра обойдется ~ в $9 млн. Плюс стоимость аппаратуры (Exadata) ~$1,5 млн.

Удачной альтернативой дорогостоящим параллельным СУБД в области больших данных являются свободно распространяемые разработки с открытым исходным кодом Hadoop [6] и Spark [7]. Обе системы высокопроизводительны, хорошо масштабируются, и их требо-

вания к аппаратной платформе весьма скромные. Это делает Hadoop и Spark перспективными системами для аналитической обработки больших массивов данных. Но они обладают средним качеством документации, не сертифицированы ФСТЭК и, главное, являются зарубежными системами. Согласно постановлению Правительства РФ № 1236 от 16 ноября 2015 г. установлен запрет на доп уск программного обеспечения, происходящего из иностранных государств, для обеспечения государственных и муниципальных нужд.

Чтобы не оказаться в догоняющей позиции, новые отечественные СУБД следует создавать на базе готовых СУБД с открытым кодом и свободной лицензией, поддерживаемых международным сообществом [8]. Этому требованию удовлетворяет открытая версия отечественной разработки Postgres Pro [9], сертифицированная ФСТЭК. Но она – одноузловая, а потому – недостаточно производительная.

Типовая архитектура реляционной СУБД по Стоунбрейкеру и Хелерстейну [10] включает в себя 5 главных компонентов (Рис. 1).

1. Менеджер коммуникации с клиентом, включающий протоколы обмена информацией для локальных и удаленных клиентов.

2. Менеджер управления процессами, выполняющий функции диспетчера и планировщика обработки.

3. Менеджер управления транзакциями – управление доступом, блокировкой, журналированием и буфером данных.

4. Общие компоненты и утилиты: пакетные утилиты, службы репликации и загрузки, утилиты для администрирования и мониторинга, менеджеры каталогов и памяти.

5. Процессор реляционных запросов. В качестве такового в данной работе применяется сервер MySQL. Несмотря на то, что MySQL является самодостаточной СУБД и полностью соответствует типовой архитектуре, все ее компоненты, показанные на Рис. 1, ориентированы на работу в рамках одного потока (процессорного ядра). Она не может применяться для параллельной обработки без существенной модификации. Поэтому из MySQL мы используем только реляционный процессор для реализации отдельных операций регулярного плана (см. ниже) на выделенных процессорных ядрах. Все другие компоненты (п. 1-4) пришлось разрабатывать отдельно для каждой рассматриваемой в этой статье архитектуры.

Для консервативных СУБД наиболее важен случай обработки потока запросов, транслируемых к схеме

$$\text{СЕЛЕКЦИЯ } (\sigma) - \text{ПРОЕКЦИЯ } (\pi) - \text{СОЕДИНЕНИЕ } (\sigma_\theta (R \times S)).$$

Здесь < x > – декартово произведение. Селектирование в операции соединения ведется

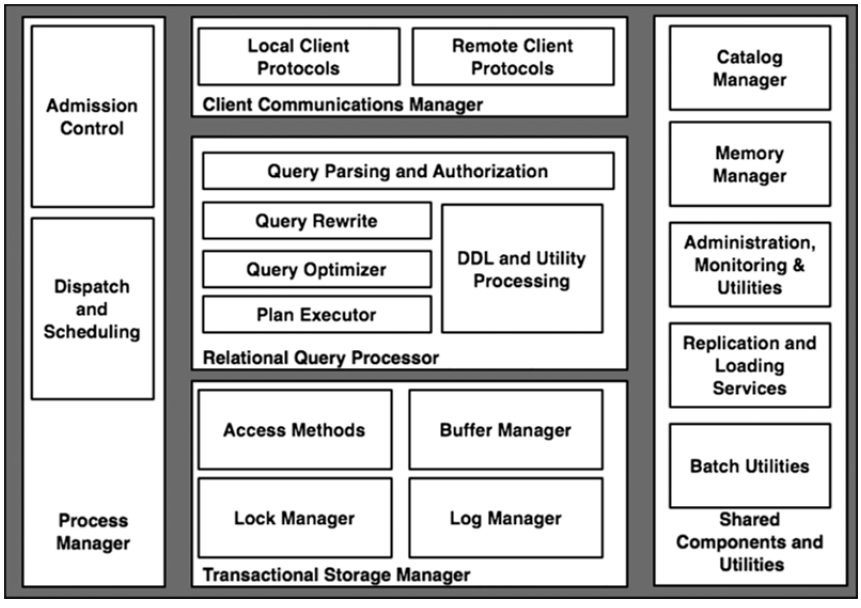

Рис. 1. Типичная архитектура реляционной СУБД

по θ-соответствию кортежей отношений R и S. Разработку параллельной СУБД желательно провести из условия реализации потоково-конвейерного способа обработки запросов. Выполнить такое условие не просто, ибо оно предполагает идеальную сбалансированность всех звеньев конвейера. Но если допустить достижимость приемлемой балансировки, то правомерен выбор регулярного плана обработки запросов Рис. 2 [11]. Особенности его реализации были уже рассмотрены в [12], но вновь отметить их в данной статье не будет лишним.

В процессе претрансляции SQL-запросов к регулярному плану формируются подзапросы (*select-project*), (*join*) и *sort* (выполняет операции агрегации (SUM(), AVG(), MAX(), MIN() и др.) и сортировки результата). При использовании стратегии «множество узлов кластера – на один запрос» база данных оказывается распределенной по узлам. Получение любого промежуточного R_i' и любого временного R_{Bj} отношений происходит параллельно на процессорах IO и JOIN. При этом теоретически возможно совмещение обоих процессов, если за время предварительной обработки (селекции с проекцией) исходного отношения R_i успевает сформироваться отношение $R_{B(i-2)}$, что является основой реализации сбалансированного конвейера с приемлемой продолжительностью его этапов.

Но ранее созданные исследовательские версии Clusterix-подобных систем [12-14] были недостаточно эффективны. Надо было искать пути повышения их эффективности. Задачей данной работы является анализ возможностей реализации экономичных консервативных СУБД повышенных объемов, сравнимых по эффективности (по критерию производительность/стоимость) с системой Spark при обработке потока запросов к БД объемом в сотни GB и более на сравнительно недорогих кластерных платформах с использованием регулярного плана обработки запросов, применением средств MySQL и GPU-акселераторов на исполнительном уровне. MySQL позволяет использовать различные «движки» и имеет систему расширений [15]. Эти особенности упрощают и ускоряют разработку системы в сравнении с использованием PostgreSQL.

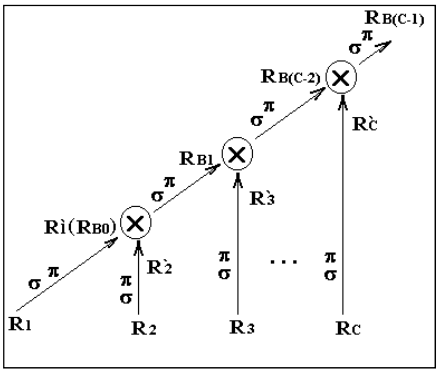

Рис. 2. Регулярный план

1. Принятые ограничения

Они диктуются требованием экономичности.

1. Аппаратной платформой исследуемых СУБД являются вычислительные кластеры, собранные фирмами из поставляемых комплектующих.

2. SMP-узлы кластера – 2-процессорные, оснащенные инструментальной СУБД MySQL и операционной системой семества Linux/Windows.

3. Процессоры в узлах – серийные с числом процессорных ядер не более 8.

4. Допускается подключение к узлам по шине PCI-e GPU-ускорителей с числом ядер не более 512.

5. Сеть связи между узлами – GigabitEthernet (10GigabitEthernet/Infiniband – по возможности).

6. Дисковая подсистема – SATA (SAS – по возможности).

7. Объем оперативной памяти в узле – не более 512 GB.

8. Хешированная база данных полностью размещается в совокупной оперативной памяти всех узлов кластера.

9. Рассматриваемые СУБД являются многопользовательскими системами с пакетной обработкой запросов.

Соответственно все исследовательские эксперименты были проведены на платформе GPU-кластера, состоящего из 7 узлов. Параметры узлов: 2 six-core E5-2640 CPU/2,5GHz/DDR3 128GB; 2 448-core GPU Tesla C-2075/1,15GHz/GDDR5 6GB (на Mgm GPU отсутствуют). Дисковая подсистема узла – RAID 10 из 4 WD1000 DHTZ/ 1TB суммарным объемом (за вычетом «зеркала») 2 TB. Операционная си-

стема – Windows Server 2012 R2. Интерконнект между узлами – GigabitEthernet с 24-портовым коммутатором SSE G24-TG4. Объемы БД – 120 GB. Представительский тест (ПТ) – конкатенация 6 перестановок TPC-H Throughput Test без операций записи.

В эксперименте со Spark БД была представлена в виде структурированных текстовых файлов и равномерно распределена по 6 исполнительным узлам. Доступ к данным был реализован с помощью Hadoop (HDFS). Балансировка нагрузки производилась модулем YARN, также входящим в состав Hadoop. Обработка запросов выполнялась Spark в конфигурации «worker на ядро» (итого 6x12=72 worker'a на кластер). Запросы запускались в работу без каких-либо изменений и оптимизаций. За работу с sql-запросами отвечало расширение Spark spark-sql, которое выполняло разбор и оптимизацию исходного запроса.

Сравнительная оценка производительности Clusterix-подобных архитектур и Spark проводится по двум показателям: 1) суммарное время обработки пакета запросов T, 2) время задержки $t_{зд}$ для каждого запроса от момента его поступления в систему до момента отправки ответа $t_{зд} = t_{обр} + t_{ожд}$. Здесь: $t_{обр}$ – время обработки запроса сервером, $t_{ожд}$ – время простоя запроса в очереди запросов сервера. Полученные данные используются для подсчета математического ожидания $M(t_{зд})=\left(\sum_{k=1}^{n} t_{зд}^{k}\right)/n$ и среднеквадратического отклонения $\sigma(t_{зд})=\sqrt{M[t_{зд}-M(t_{зд})]^2}$, k – номер запроса ПТ, n – число запросов в ПТ.

Поскольку обозначена эталонная СУБД, все эксперименты проводятся на одном тесте и на одной платформе, значения M и σ немаловажны для пользователя, а сравниваемые архитектуры относятся к классу open source, вводится

Определение 1. Сравнительная эффективность Clusterix-подобных систем определяется как кортеж $<\Delta T, \Delta M, \Delta\sigma>$, где $\Delta T = T_{Sp}/T_{Cl}$, $\Delta M = M_{Sp}/M_{Cl}$, $\Delta\sigma = \sigma_{Sp}/\sigma_{Cl}$.

2. Методология, использованная при решении задачи

За основу проведенных исследований была принята методология КМС – конструктивного моделирования систем [16]. Кардинальные вопросы синтеза в условиях *неполноты информации*:

– ГДЕ (в какой области некоторого пространства) искать нужное решение?

– КАК (какими методами) организовать такой поиск?

– ПОЧЕМУ именно там и так?

Методологическую основу КМС составляют следующие положения.

1. Предполагается, что синтезируемый объект моделирует поведение некоторой гипотетической системы – нечто единого целого, бесконечно познаваемого и объясняемого, заданного своим оператором назначения. Моделирование такой системы трактуется как ***S***-моделирование процесса синтеза (***S*** – от Synthesis). Под процессом в кибернетике понимается последовательная смена состояний некоторого объекта. Поэтому разрабатываемая модель – не статичное образование, а динамически развивающаяся (*эволюционирующая*) система, каждому состоянию которой отвечает определенное качество моделирования. Развитие прекращается по получении требуемого качества. В итоге получаем искомый *конструктивный метод*. Таково обоснование принятого названия – *конструктивное моделирование систем*.

2. Свойства, которыми должно обладать устройство, чтобы ***S***-моделирование было достаточно эффективным, могут быть выявлены в динамике ***S***-моделирования в виде *постулатов, утверждающих достаточно проверенные идеи.* Процесс ***S***-моделирования рассматривается как многошаговый итеративный процесс, в котором взаимодополнительно проявляются как объяснительно-содержательные посылки (постулаты как элементы теории), так и сам конструктивный метод (реализация приемлемой итерации ***S***-модели). Система постулатов должна быть открытой для корректив. Введение постулатов целесообразно, только если разработка конструктивного метода на их основе показывает его перспективность для своего времени, а сам метод не укладывается в рамки существующей теории.

3. Конечной целью ***S***-моделирования является разработка теоретически обоснованного конструктивного метода, т.е. процедуры синте-

за. Формально, процесс ***S***-моделирования включает 2 этапа – *внешнее моделирование* (постулирование математической ***S***-модели как релевантного описания – фреймового, логического, алгебраического или др. – ориентированной последовательности областей полного множества решений – ответы на вопросы: ГДЕ? и ПОЧЕМУ?), и *внутреннее* (итеративное исследование найденной ***S***-модели с целью разработки конструктивного метода – ответ на вопрос: КАК?).

4. Различают ***S***-модели: у*нитарные* (***US***-модели) и *иерархические* (***IS***-модели). ***US***-модель – единый абстрактный образ (единственная область поиска), например, локальная область некоторого метрического пространства. Предпочтительность такого описания систем несомненна. ***IS***-модель – множество представлений иерархической системы. Строится, когда единый абстрактный образ системы найти не удается. Системное уравновешивание значений модельных показателей всех уровней иерархии достигается в процессе внутреннего моделирования.

Большие системы, как правило, иерархические – ***IS***-модели. Процесс ***IS***-моделирования не должен занимать слишком много времени, как это свойственно естественной эволюции. Поэтому в таком процессе должна быть найдена математическая (внешняя) модель как минимальный набор состояний (областей) в пространстве всевозможных состояний ***IS***-модели, переходы между которыми образуют *кратчайший путь* получения искомого результата. Алгоритмическая и программная разработка каждого состояния является предметом внутреннего моделирования.

К числу ***IS***-моделей относятся и СУБД с уровнями иерархии: *select-project*, *join*, *sort*, динамическая сегментация отношений, их индексация, сетевой и др. В данном случае оператор назначения гипотетической системы задан условием получения высокой эффективности обработки запросов (по введенному ранее сравнительному критерию в виде кортежа $<\Delta T, \Delta M, \Delta\sigma>$) при минимальной стоимости системы, определенной ранее сформулированными ограничениями. Состояние ***IS***-модели – это архитектура программной системы как совокупность взаимодействующих программных модулей. Его название будем ассоциировать с некоторым характерным признаком, а полную программную разработку будем называть полным состоянием.

IS-моделирование никогда не проводится на «пустом месте». Начальное состояние ***IS***-модели заведомо определено тем или иным способом. От пространства полных состояний можно перейти к пространству параметров. Под параметром будем понимать среднее время обработки одного запроса ПТ на том или ином уровне. Для заданной платформы существует однозначное отображение пространства полных состояний в пространство параметров (вопрос о взаимной однозначности оставляем открытым), в котором и будем вести рассмотрение. По аналогии с принятым в синергетике [17], для каждого полного состояния будем выделять так называемый «параметр порядка», необходимость снижения величины которого определяет переход к соседнему состоянию.

В качестве такового принимается параметр, имеющий максимальное значение для данного полного состояния. Но функционирование всех уровней в большой системе взаимосвязано (принцип единства системы). Поэтому снижение влияния «параметров порядка» на производительность системы неизбежно влечет изменение влияния и других уровней. Число итераций обычно сравнительно невелико, если критерий качества конечного решения приемлемо задан (в данном случае – получение эффективности, сравнимой с эффективностью системы Spark).

3. Принятые постулаты

В процессе проведенного ***IS***-моделирования сформулирована система постулатов как декларация целесообразных направлений разработок искомых моделей.

Постулат 1. Решение поставленной задачи должна обеспечить эволюция Clusterix-подобных СУБД от начальной реализации принципов гибридной технологии (см. ниже принятое начальное состояние ***IS***-модели).

Постулат 2. Поиск очередных состояний (итераций) ***IS***-модели Clusterix-подобных СУБД следует вести на пути замены стратегии «ядро на одно отношение», принятой для ее начального

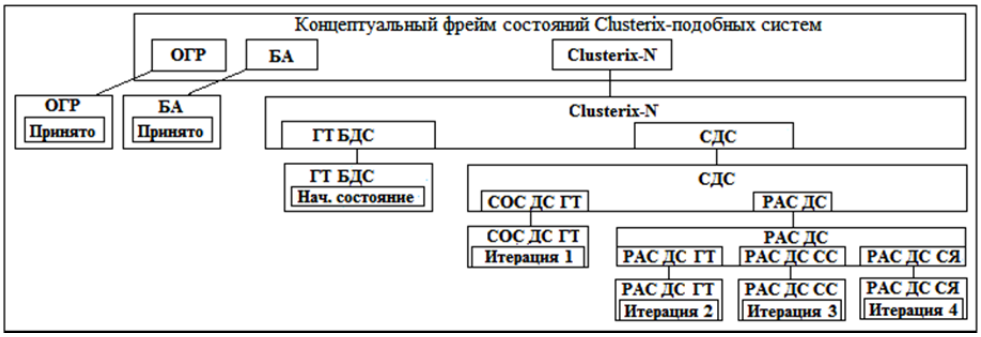

Рис. 3. Внешняя фреймовая модель процесса синтеза Clusterix-подобных систем

ОГР	–	фрейм принятых ограничений
БА	–	фрейм принятой блочной архитектуры в составе 5 программных блоков: IO (модуль доступа к данным БД, выполняет подзапросы типа *select-project*), JOIN (модуль обработки подзапросов типа *join*), Mgm (модуль управления), SORT (модуль конечной обработки запроса), HASH (реализует динамическую сегментацию отношений; в начальном состоянии он отсутствует)
Clusterix-N (N – от New)	–	фрейм развития Clusterix-подобных систем
ГТ БДС	–	фрейм начального состояния IS-модели (переход к гибридной технологии Clusterix-подобных систем без динамической сегментации отношений)
СДС	–	фрейм Clusterix-N с динамической сегментацией отношений
СОС ДС ГТ	–	фрейм СДС-систем с сосредоточенной динамической сегментацией отношений в рамках гибридной технологии (первая итерация IS-моделирования)
РАС ДС	–	фрейм СДС-систем с распределенной динамической сегментацией
РАС ДС ГТ	–	фрейм (РАС ДС)-систем, реализованных по гибридной технологии (вторая итерация IS-моделирования)
РАС ДС СС	–	фрейм (РАС ДС)-систем в конфигурации «совмещенная симметрия» (третья итерация IS-моделирования)
РАС ДС СЯ	–	фрейм (РАС ДС)-систем в конфигурации «совмещенное ядро» (четвертая итерация IS-моделирования)

состояния, на стратегию «группа узлов (ядер) на одно отношение». Это необходимо для обеспечения надежной работы эффективной системы при значительных объемах баз данных и требует динамической сегментации отношений, которая может быть как сосредоточенной, так и распределенной.

Постулат 3. Внутреннее ***IS-***моделирование Clusterix-подобных систем следует проводить в направлениях, определенных внешней (математической) фреймовой моделью процесса синтеза, показанной на Рис. 3.

4. Характеристика начального состояния

В первых Clusterix-подобных системах для ускорения операций *join* было использовано динамическое сегментирование промежуточных и временных отношений по мере формирования отдельных записей R_i' и R_{Bj}. Оно занимало достаточно много времени и с ростом числа узлов могло приводить к сбоям в работе системы. В работе [12] было показано, что повысить производительность можно переходом к архитектуре Clusterix-N путем отказа от принципа «однородности» (характерного для оптимальной Clusterix-конфигурации «совмещенная симметрия» [13]) в пользу «гибридности», подразумевающей разделение кластера на две различные части – блоки IO и JOIN – с независимой вариацией числа узлов в каждом блоке. Это и явилось причиной сделанного выбора начального состояния.

База данных хешировалась на уровне узлов IO. В них реализована стратегия «отношение на ядро». На уровне узлов Join использовалась стратегия «запрос на ядро» (реализуемость такой стратегии средствами MySQL показана в работе [18]), что позволяло исключить динамическое сегментирование промежуточных и временных отношений и выполнять соединение в виде единой процедуры R1′ *join* (*join* R2′(*join* R3′(…))) …) по каждому запросу. Это гораздо быстрее ее последовательного выполнения и приводит к значительному росту эффективности в сравнении с Clusterix.

Детальная программная разработка начального состояния позволила создать своеобразные

«модули-заготовки», которые в дальнейшем модифицировались для каждого нового состояния. Наличие таких «заготовок» существенно облегчило проведение последующих итераций, где они соответственно модифицировались. Это – подсистемы сбора статистики, визуализации и журналирования; модуль сетевого взаимодействия; драйвер СУБД; модуль MGM как ядро системы; модули IO, JOIN и SORT; способ претрансляции запроса к регулярному плану; настройка MySQL на максимальную загрузку всех процессорных ядер узла.

Эффективность начального состояния оказалась существенно ниже, чем у Spark. Кроме того, уже при объемах баз данных < 100GB большой суммарный объем промежуточных отношений по некоторым запросам теста TPC-H приводил к перегрузке оперативной памяти узлов JOIN и, как следствие, – к потере работоспособности СУБД. *Ненадежность* – параметр порядка для начального состояния.

5. Первая итерация IS-моделирования

Надежная работа с БД объемом в сотни GB и более требует перехода на стратегию «множество ядер в каждом блоке на одно отношение», для реализации которой необходима динамическая сегментация. Поэтому она восстановлена в первой итерации Clusterix-N [14], но (в отличие от Clusterix [13]) с передачей сегментов в целом. Ее осуществляет модуль HASH на выделенном узле с GPU-ускорителями, распределяя данные по всем процессорным ядрам уровня JOIN. Хеширование выполняется с использованием алгоритма деления [19]. Результат хеширования помещается в буфер отправки по ядрам (для каждого ядра в узлах JOIN модуль HASH формирует буфер в своей памяти). Отправка данных происходит по готовности операции хеширования.

В результате внесенных изменений программа Clusterix-N теперь состоит из 5 модулей: MGM, IO, JOIN, HASH, SORT. Как и ранее, БД распределена по узлам IO. Модули IO и JOIN реализуют стратегию «группа узлов на отношение». Конфигурация кластера при проведении эксперимента для первой итерации: 2 узла IO, 3 узла JOIN, 1 узел HASH и 1 узел MGM, совмещающий модули MGM и SORT.

Принципиальной особенностью этой итерации (и дальнейших) является конвейерно-циклическое выполнение операций *select-project* и *join* по каждому запросу (наличие внутренней конвейеризации вплюс ко внешней). Но и теперь Clusterix-N остается неконкурентоспособной. Результаты экспериментов для времени обработки ПТ таковы: Clusterix-N – 19,7 час; Spark – 4,5 час. Они явно не в пользу Clusterix-N.

Модуль JOIN, как и IO, оснащен СУБД MySQL. Он тоже претерпел ряд изменений. Теперь в зависимости от конфигурации системы, модуль принимает задание и данные не только от MGM, но и от модуля HASH, производит загрузку данных в MySQL и запускает операцию *join*. Загрузка данных контролируется специальным загрузчиком и производится командой LOAD FILE после полного получения всех данных для загружаемого отношения. Но, в связи с реализацией распределённой обработки, теперь для каждого ядра предусматривается свой набор отношений, что допускает параллельную загрузку данных и параллельное выполнение операции *join*. Введена очередь запросов, поскольку данные для их обработки могут быть получены заранее.

Администратор выполняет настройку перед началом работы системы. Запросы в очереди могут выполняться одним из методов: параллельный, интегрированный или последовательный *join*. Параллельный – запускается на всех свободных ядрах вычислительного узла и эффективно работает с большим массивом данных. Последовательный – выполняет *join* один за другим на одном ядре и позволяет обрабатывать сразу несколько запросов в небольшой БД. Интегрированный – выполняет сразу все необходимые операция *join* и требует значительного объема памяти в узле. Запуском запросов из очереди и контролем их выполнения занимается менеджер JOIN, который загружает метод *join* и передает ему управление. Результат *join*-обработки передается в модуль HASH сразу после его выполнения. Все промежуточные отношения удаляются по завершении обработки подзапроса. Поскольку наше рассмотрение ориентировано на БД повышенных объемов, в статье рассматривается только метод параллельного *join*.

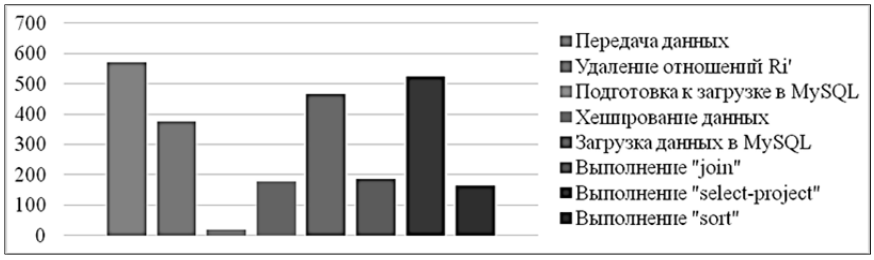

Рис. 4. Среднее время обработки одного запроса ПТ по уровням (1 итерация)

Как и ранее [7], модуль SORT использует свой MySQL. Аналогично модулю JOIN, он получает задание и данные от УПР MGM и BUF MGM, производит загрузку данных в MySQL, запускает операцию *sort*. Результат работы модуля SORT размещается в новом отношении, которое передается по готовности в BUF MGM и от него – пользователю. Работа модуля организуется по стратегии «запрос на ядро».

Выявленный в итоге первой итерации параметр порядка определен гистограммами Рис. 4 (по оси ординат – время в сек.). Это – временной вклад *сетевого уровня*.

6. Вторая итерация IS-моделирования

Основная идея, положенная в основу второй итерации и позволявшая надеяться на успех, состоит в реализации операций динамического сегментирования промежуточных/временных отношений (хеширования) в модулях IO и JOIN с передачей хешированных данных напрямую между исполнительными узлами (минуя MGM). Отказ от выделенного хеширующего узла HASH и перенос его функционала на исполнительные узлы (с применением в них для хеширования GPU-ускорителей) должен ускорить процесс передачи данных (из-за уменьшения объема передаваемых данных). Реализация хеширования, разработанная для модуля HASH, была адаптирована и перенесена в программные модули IO и JOIN. Организация режима работы системы с прямой передачей данных между исполнительными узлами потребовала изменений в модуле MGM.

Модуль IO выполняет операцию *select-project* для одного отношения параллельно на множестве доступных процессорных ядер с получением набора блоков результата. Эти блоки подвергаются хешированию с ускорением на GPU и передаются сразу в определенные узлы JOIN. Поблочная выборка позволяет совместить во времени 3 операции: после формирования одного блока запускается операция выборки следующего, результат передается в очередь на хеширование, а хешированные блоки поступают в очередь на передачу.

Модуль JOIN полностью повторяет алгоритм его работы в первой итерации. Отличие заключается лишь в обработке результата *join*. Теперь он хешируется на GPU и передается в узлы JOIN (для выполнения следующей операции *join*) или SORT с совмещением операций аналогично IO. Модуль SORT использует стратегию «запрос на ядро» и передает результат в MGM. Единственное изменение в его работе – это получение данных от узлов JOIN, а не из BUF MGM.

Экспериментальное исследование программно реализованной новой версии Clusterix-N производилось в конфигурации GPU-кластера: 2 узла IO, 4 узла JOIN, и 1 узел MGM, совмещающий модули MGM и SORT. БД распределена по узлам IO. Анализ результатов эксперимента показал, что время передач по сети уменьшилось ~ в 3 раза, операции *join* ускорились ~ в 1,5 раза (за счет добавления еще одного узла и уменьшения объема данных в каждом узле). И все же: время выполнения ПТ – 14,5 часа, т.е. общее время обработки ПТ сократилось всего на ~26% по сравнению с предыдущей модификацией Clusterix-N. Этого явно недостаточно для того, чтобы говорить о возможной конкуренции со Spark.

Параметр порядка для итерации 2 определяют гистограммы Рис. 5. В данном случае – это время выполнения операций на уровне *select-project.*

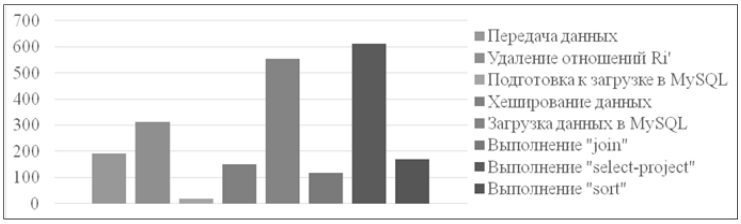

Рис. 5. Среднее время обработки запроса по уровням (2 итерация)

7. Третья итерация IS-моделирования

Наиболее простой способ ускорения операций на указанном уровне – уменьшить объем данных, обрабатываемых в одном узле, что требует увеличения числа узлов каждого блока. При неизменном общем числе узлов кластера нужного эффекта можно добиться возвратом (на новом витке «спирали») к конфигурации «совмещенная симметрия» [13], которая предполагает размещение на одном узле сразу двух модулей: IO и JOIN.

Эксперимент в этой конфигурации проводился при следующем распределении узлов GPU-кластера: 6 узлов с модулями IO и JOIN, 1 узел MGM, совмещающий модули MGM и SORT. БД распределена по 6 узлам. Каждому модулю выделен один CPU (6 ядер) и один GPU-ускоритель. На всех узлах (кроме MGM) функционирует сразу две СУБД MySQL: одна – для IO, другая – для JOIN. Запуск двух СУБД MySQL обусловлен архитектурой SMP-узлов и разной конфигурацией СУБД: для IO конфигурация нацелена на оптимизацию работы запросов *select-project*, для JOIN – на оптимизацию работы «движка» MEMORY и операций *join*. Полученные гистограммы показаны на Рис. 6.

Результаты эксперимента в сравнении со Spark представлены в Табл. 1. То, что Clusterix-N значительно уступает системе Spark по значениям M и σ, немаловажно для пользователя.

Сравнение усредненных на множестве запросов ПТ времен выполнения отдельных операций для Clusterix-N представлено в Табл. 2. Динамику процессов в этой итерации иллюстрирует Рис. 7. Как следует из Табл. 2 и гистограмм Рис. 6, в конфигурации «совмещенная симметрия» самой длительной операцией осталась *загрузка данных в MySQL*. Это и есть пресловутый «параметр порядка» для третьей итерации.

8. Четвертая итерация IS-моделирования

Загрузку данных в MySQL для модулей JOIN можно ускорить увеличением числа ядер, на которых реализуются эти модули. Можно заметить (Рис. 7), что модули IO далеко не всегда заняты обработкой запросов, ибо один из CPU в каждом узле простаивает продолжительное время. Не лучше ли будет последовательная реализация операций *select-project* и *join* на одном ядре

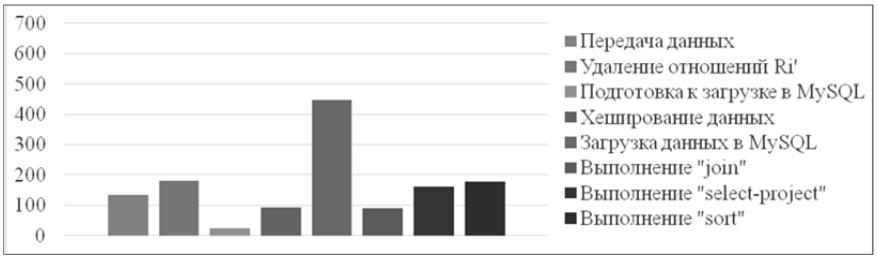

Рис.6. Среднее время обработки запроса ПТ (3 итерация)

Табл. 1. Результаты для конфигурации «совмещенная симметрия») в сравнении со Spark

	Clusterix- N	Spark	Отношение Spark / Clusterix-N
T, мин	455,8	260,6	0,57
M, мин	14,3	3,1	0,22
σ, мин	15,7	0,9	0,06

с загрузкой этими операциями всех ядер кластера? В таком случае модули IO будут работать без простоев, а операции загрузки в MySQL будут значительно ускорены, что должно снизить значения M и σ. Но не снизит ли эффективность такое нарушение внешней конвейеризации?

Табл. 2. Среднее время выполнения отдельных операций в Clusterix-N

	Clusterix-N Итерация 1 α, сек	Clusterix-N Итерация 2 β, сек	Clusterix-N Итерация 3 γ, сек	$\frac{\alpha}{\beta}$	$\frac{\beta}{\gamma}$	$\frac{\alpha}{\gamma}$
Передача данных	569,77	190,34	133,62	2,99	1,42	4,26
Удаление отношений R_i',$Rв_j$	375,27	308,17	180,03	1,22	1,71	2,08
Подготовка к загрузке	18,90	18,19	22,40	1,04	0,81	0,84
Хеширование данных	177,03	148,93	93,61	1,19	1,59	1,89
Загрузка данных в MySQL	466,64	562,10	447,28	0,83	1,26	1,04
Выполнение «*join*»	186,09	122,43	89,09	1,52	1,37	2,09
Выполнение «*select-project*»	522,73	685,37	159,86	0,76	4,29	3,27
Выполнение «*sort*»	162,61	164,00	178,42	0,99	0,92	0,91

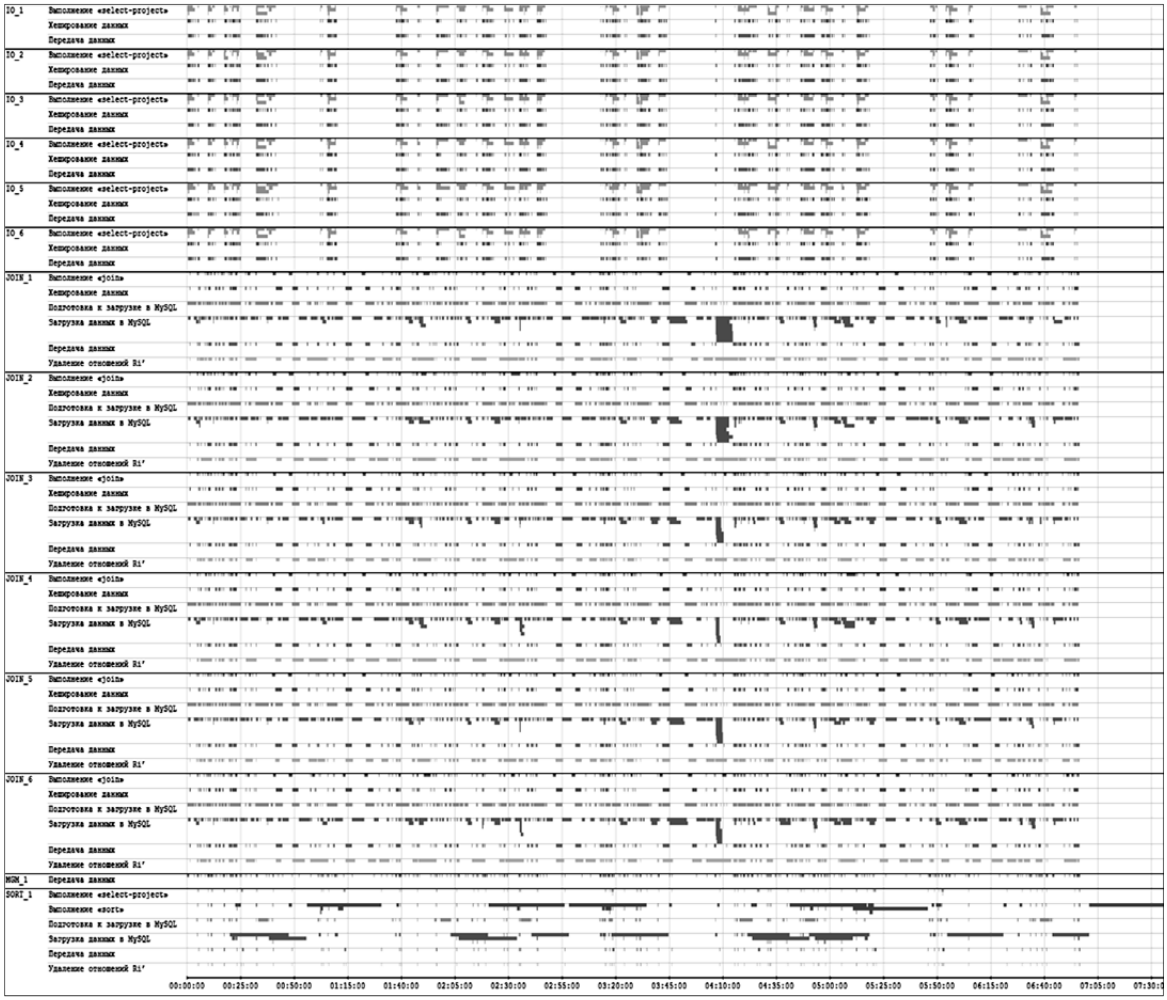

Рис.7. Визуализация выполнения ПТ (итерация 3)

Для получения ответа на этот вопрос выполнено предварительное исследование при минимальных доработках системы. В каждом узле используется по одному GPU для целей хеширования результатов работы модулей IO и JOIN. Операции модуля SORT ускорены за счет смены «движка» MySQL с MyISAM на MEMORY.

Экспериментально полученные гистограммы представлены на Рис. 8. Им отвечают данные Табл. 3 и Табл. 4. Они подтверждают сделанные прогнозы и опасения. Несмотря на серьезное ускорение операции загрузки в MySQL, уменьшение M и σ, время обработки ПТ увеличилось~ на 23% в сравнении с итерацией 3.

Такова плата за частичное нарушение конвейеризации. Несколько поправить положение можно дальнейшим ускорением *select-project* и ряда других операций. При неизменной платформе, эффективность итерации 4 можно повысить, во-первых, переходом в IO от хеширования БД по узлам к хешированию по ядрам. Это должно значительно ускорить *select-project*. Во-вторых, – переходом на более совершенную версию MySQL 8.0. Можно ожидать, что ее применение ускорит выполнение и ряда других операций. Программная реализация таких переходов была связана с доработками драйвера СУБД и модуля IO.

Результаты проведенного эксперимента иллюстрируют гистограммы Рис. 9 и данные Табл. 5 и Табл. 6. Теперь оценки для Clusterix-N и Spark сравнимы в бóльшей степени.

9. Целесообраность организации работ со сжатыми базами данных

При ограниченном числе узлов, база данных значительных объемов может не разместиться в оперативной памяти кластера. Поэтому полезно рассмотреть возможную организацию работ со сжатыми базами данных. Покажем, что при

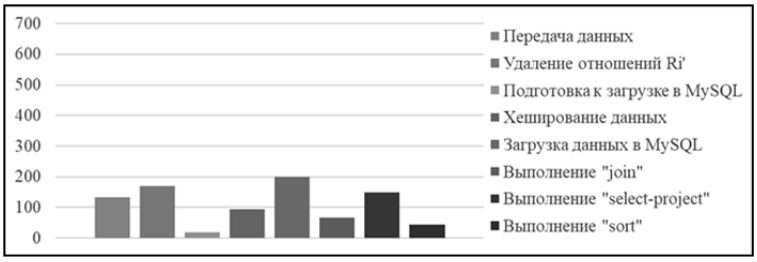

Рис. 8. Гистограммы для начала итерации 4

Табл. 3. Сравнительная оценка ускорений для итераций 3 и н.4

	Итерация 3, сек	Итерация н.4, сек	Отношение и.3 /н.4
Передача данных	133,62	133,36	1,00
Удаление отношений $Ri',Rв_j$	180,03	170,47	1,06
Подготовка к загрузке в MySQL	22,40	17,80	1,26
Хеширование данных	93,61	94,62	0,99
Загрузка данных в MySQL	447,28	198,75	2,25
Выполнение "join"	89,09	66,83	1,33
Выполнение "select-project"	159,86	148,17	1,08
Выполнение "sort"	178,42	43,03	4,15

Табл. 4. Сравнительные оценки по T, M и σ итераций 3 и н.4

	Итерация и.3, мин	Итерация н.4, мин	Отношение и.3/н.4
T	455,8	560,6	0,81
M	14,3	8,2	1,74
σ	15,7	6,4	2,45

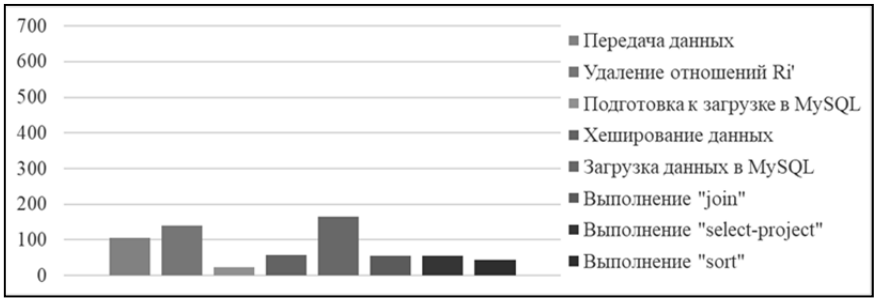

Рис. 9. Гистограммы для окончательной версии итерации 4

Табл. 5. Оценка ускорений при переходе от начала итерации 4 к ее концу

	Итерация н.4, сек	Итерация и.4, сек	Отношение н.4 /и.4
Передача данных	133,36	106,12	1,26
Удаление отношений Ri', Rвj	170,47	140,91	1,21
Подготовка к загрузке в MySQL	17,80	24,15	0,74
Хеширование данных	94,62	56,43	1,68
Загрузка данных в MySQL	198,75	164,01	1,21
Выполнение "join"	66,83	54,77	1,22
Выполнение "select-project"	148,17	55,40	2,67
Выполнение "sort"	43,03	42,90	1,00

Табл. 6. Окончательное сравнение по T, *M* и σ итерации 4 и Sparc, $V_{БД}$ = 120 GB

	Итерация 4, мин	Spark, мин	Отношение Spark /и.4
T	403,8	260,6	0,65
M	6,3	3,1	0,49
σ	5,6	0,9	0,16

хранении сжатой БД в оперативной памяти узлов применение GPU-ускорения операции *select* может дать ощутимый эффект. При этом несколько сместим акценты работы [20].

Как известно, использование GPU позволяет многократно снизить время выполнения отдельных операций [21, 22]. Но при значительных $V_{БД}$ необходимость обмена данными CPU ↔ GPU существенно снижает производительность системы в целом. Скорость передачи данных по шине PCI-e значительно ниже, чем скорость обмена с оперативной памятью. Так, скорость чтения/записи для 3-канальной оперативной памяти типа DDR3-1600 составляет 38,4 GB/s, в то время как для шины PCI-e 2.0x16 – 6,4 GB/s. Именно по этой причине достигнутый в [23] рост производительности сервера БД от использования GPU при обработке достаточно простых одиночных запросов не превысил 40%.

Различают СУБД, ориентированные на хранение данных по строкам и по столбцам. Лучшие показатели по сжатию имеют СУБД второго типа. Этот вопрос детально исследован в [24]. Сжатая база данных, разделена на блоки. Под блоком данных далее понимается часть сжатого столбца (либо набор «коротких» столбцов) с объемом разжатых данных, равным объему буфера разжатых данных GPU. Операции *select* и *динамической сегментации* отношений выполняют графические ускорители, а узловые CPU на этапе IO занимаются проецированием, продвижением очереди подзапросов, формированием и передачей сжатых блоков в GPU, получением от них результатов.

Алгоритм подготовки данных для сжатия следующий.

1. Найти самое «длинное» поле (длиной RS) в обрабатываемом отношении R.

2. Найти в соответствующем столбце (столбцах) количество записей RC, которое гарантированно умещается в отведенной памяти, RC=[BS/RS], где BS – объем памяти, отведенной для разжатых данных в графическом ускорителе.

3. Выдавать из отношения R данные по столбцам для сжатия с шагом RC.

Проведем упрощенное рассмотрение вопроса. Для каждой таблицы, участвующей в обработке, блоком IO проецируется набор колонок, необходимых для выполнения операций *select*. В GPU по-блочно передаются колонки одной таблицы, над ними производятся необходимые операции, включая селектирование и сегментацию. Результат возвращается в хост-память. И так – для всех отношений, участвующих в обработке каждого запроса. Операции *select* существенно уменьшают объем данных. Поэтому результат этой операции перед отправкой в CPU не сжимается.

В качестве инструментальной была выбрана СУБД MySQL 5.6. Эксперимент проводился на базе вычислительного узла с следующими характеристиками: Quad-core Intel Core i5-4670K CPU/2,5 GHz/24GB RAM (DDR3-1600 в двухканальном режиме), 64-битная ОС Windows 8, дисковая подсистема узла – SSD SV300S37A/120GB с пропускной способностью 450 MB/s, GPU – Nvidia GTX 770 (с объемом памяти 2GB GDDR5). Эксперимент преследовал цель сравнения времен, затрачиваемых на простое копирование, с суммой времен, необходимых для копирования данных, сжатых по алгоритму RLE (Run Length Encoding) [25], и их разжатия в GPU. Наибольшая эффективность передачи данных была достигнута для коэффициента сжатия K = 4-5. При бóльшем сжатии увеличивается время разжатия данных, которое может превысить время обычного копирования.

В Табл. 7 показан процент увеличения эффективности передачи сжатых данных (K = 4 и 5) с последующим разжатием по сравнению с простым копированием. Как видно из таблицы, при передаче небольшого объема данных простое копирование оказывается быстрее. При объемах >12 Мбайт предварительное сжатие дает ускорение в окрестности 30% (примерно в 1,5 раза).

Качественная оценка эффективности перехода на сжатые БД с GPU-ускорением операций *select* была получена следующим образом. Согласно регулярному плану обработки, для 14 отобранных запросов теста TPC-H были сформированы загрузочные модули «*select – project»*. При этом все операции «*project»* были «опущены вниз». Объем тестовой БД составил 1GB. Она предварительно загружалась в оперативную память. Платформа эксперимента осталась прежней.

Табл. 7. Изменение эффективности передачи данных

Размер (Мбайт)	Копирование, мс	Копирование и восстановление, мс		Ускорение, %	
		K = 4	K = 5	K = 4	K = 5
0,38	1	4	4	-300%	-300%
4,20	3	6	8	-167%	-100%
8,01	5	9	9	-80%	-80%
11,83	7	12	12	-71%	-71%
15,64	21	15	15	29%	29%
19,45	26	20	19	27%	23%
23,27	31	23	24	23%	26%
27,08	35	26	27	23%	26%
30,90	41	30	29	29%	27%
34,71	45	35	32	29%	22%
38,53	50	35	35	30%	30%
42,34	55	38	38	31%	31%
46,16	59	41	42	29%	31%
9,97	64	44	44	31%	31%
53,79	68	48	48	29%	29%
57,60	75	50	51	32%	33%
61,42	80	53	53	34%	34%

При подсчете объема данных для любого из рассматриваемых запросов определялась длина каждого поля каждой строки всех таблиц, участвующих в обработке запроса, и все полученные длины суммировались. Подсчет выполнялся средствами MySQL путем модификации запросов. Пример модификации запроса №3:

```
 -- O'
 SELECT     O_ORDERDATE,     O_SHIPPRIORITY,
O_ORDERKEY, O_CUSTKEY
 FROM ORDERS
 WHERE O_ORDERDATE < DATE '1995-03-31';
 -- O' LENGTH
 SELECT SUM(LENGTH(O_ORDERDATE),
LENGTH(O_SHIPPRIORITY), LENGTH(O_ORDERKEY),
 LENGTH(O_CUSTKEY))
 FROM ORDERS
 WHERE O_ORDERDATE < DATE '1995-03-31';
```

Здесь O' – один из подзапросов «*select – project»*, полученных после претрансляции этого запроса; O' LENGTH – запрос для подсчета объема возвращаемых данных.

Результаты измерений объемов информации до и после «*select – project»* приведены в Табл. 8. В этой таблице: $V_{общ}$ – полный объем отношений, необходимый для выполнения запроса; $(\sigma,\pi)_{\Sigma}$ – сумма объемов тех же отношений после уменьшения объема данных с использованием условий выборки из запроса. Имеем, в среднем, примерно 7-кратное уменьшение объема данных.

Таким образом, суммарные объемы передач CPU → GPU на стадии IO для сжатой БД и без сжатия соотносятся как 7:5 (превышение на 40%). Но время селектирования на CPU в десятки и более раз превышает то же время на GPU.

Заключение

В работе показано, что соответствующая архитектурная и программно-алгоритмическая разработка экономичных консервативных Clusterix-подобных СУБД класса BigData показывает результаты, сравнимые по эффективности с лучшими открытыми системами. Это подтверждает приведенная ниже сводная таблица результатов итераций 1 – 4 для $V_{БД}$=120GB. Она говорит о том, что переход от итерации 1 к итерации 4 снижает (Табл. 9): $T\sim$ в 2,8 раз, $M\sim$ в 4,6 раз, $\sigma\sim$ в 4,5 раз.

Стоимость приобретения и ввода в эксплуатацию GPU-кластера, аналогичного использованному при проведении сравнительных экспериментов, составит не более $85 000. Все версии программной системы Clusterix-N помещены в открытый доступ [26] и могут быть использованы заинтересованными организациями.

Табл. 8. Объемы данных ПТ до и после *«select-project»*

№ запроса	Объем данных для обработки, байт		Коэффициент уменьшения объема данных
	До выборки по условиям	После выборки по условиям	
	$V_{общ}$	$(\sigma, \pi)_{\Sigma}$	$V_{общ}/(\sigma, \pi)_{\Sigma}$
1	675 846 277	134 255 929	5,03
2	*137* 651 393	13 526 703	10,18
3	855 794 582	76 991 966	11,12
4	832 798 438	428 049 230	1,95
5	857 126 234	139 324 211	6,15
6	675 846 277	1 339 256	504,64
7	857 125 865	78 759 670	10,88
8	879 261 359	179 338 247	4,90
9	970 449 462	234 598 226	4,14
10	855 796 681	49 964 804	17,13
11	342 555 947	27 528 586	12,44
12	832 798 438	23 140 549	35,99
13	179 948 305	18 706 950	9,62
14	697 981 402	6 548 108	106,59
Итого	9 695 098 066	1 412 072 435	6,87

Табл. 9. Сводная таблица результатов экспериментов для итераций 1-4

		Clusterix-N	Spark	Spark /Clusterix-N
Итерация 1	***T, час***	**19,7**	**4,3**	**0,22**
	M, мин	29,0	3,1	0,11
	σ, мин	25,4	0,9	0,035
Итерация 2	***T, час***	**14,5**	**4,3**	**0,30**
	M, мин	23,2	3,1	0,13
	σ, мин	20,5	0,9	0,044
Итерация 3	***T, час***	**7,6**	**4,3**	**0,57**
	M, мин	14,3	3,1	0,22
	σ, мин	15,7	0,9	0,057
Итерация 4	***T, час***	**6,7**	**4,3**	**0,64**
	M, мин	6,3	3,1	0,49
	σ, мин	5,6	0,9	0,161

И все же по значениям T, *M* и σ СУБД Clusterix-N пока что заметно уступает системе Spark. Как следует из гистограмм Рис. 9, процессы удаления отношений и загрузки данных в MySQL остаются достаточно медленными. Есть основания полагать, что для MySQL существует принципиальное ограничение на достижимое ускорение этих процессов. Как показывает анализ динамики их выполнения в итерации 4, все ядра системы полностью заняты лишь в начальные моменты загрузки. Далее система переходит к работе на одном ядре.

Выход из положения видится в разработке специальных движков MySQL для ускорения указанных процессов. Как показано в разд. 9, заметный рост эффективности системы должен иметь место и в случае перехода к работе со сжатыми базами данных при дополнительном GPU-ускорении операций *select*. Кроме того, не следует сбрасывать со счета и возможный выигрыш от передачи по сети сжатых данных [20].

Но реализация и оценка этих возможностей – предмет дальнейших исследований.

Литература

1. E. F. Codd. Providing olap to user-analysts: an it mandate, Apr. 1993. Technical Report, E. F. Codd and Associates.
2. Microsoft. Parallel Query Processing //Resources and Tools for IT Professionals | TechNet. 2018. URL: https://technet.microsoft.com/en-us/library/ms178065(v=sql.105).aspx (дата обращения: 05.04.2018).
3. Lenovo System x3950 X6 // TPC-H Result Highlights. 2016. URL: http://www.tpc.org/3321 (дата обращения: 10.08.2018).
4. Lenovo. System x3950 X6 Rack Server //Официальный сайт Lenovo в России. 2017. URL: https://www3.lenovo.com/ru/ru/data-center/servers/mission-critical/System-x3950-X6/p/WMD00000002 (дата обращения: 15.07.2018).
5. Oracle Exadata Database Machine X7 //Oracle Россия и СНГ. 2018. URL: https://www.oracle. com/ru/engineered-systems/exadata/database-machine-x7/index.html (дата обращения: 10.08.2018).
6. EMC Education Services. Data Science and Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data // John Wiley & Sons. 432 p.
7. Xin, Reynold & Rosen, Josh & Zaharia, Matei & J. Franklin, Michael & Shenker, Scott & Stoica, Ion. (2012). Shark: SQL and Rich Analytics at Scale. Proceedings of the ACM SIGMOD International Conference on Management of Data. 10.1145/2463676.2465288.
8. Российская отрасль СУБД продвигается на «слонах» //Connect. 2017. №5-6. С.34-38.
9. Российская СУБД Postgres Pro //Postgres Professional. 2018. URL: https://postgrespro.ru/products/postgrespro (дата обращения: 03.05.2018).
10. Hellerstein J.M., Stonebraker M., Hamilton J. Architecture of a Database System //Foundations and Trends in Databases. 2007. Vol. 1. No. 2. pp. 141-259.
11. Raikhlin V.A. Simulation of Distributed Database Machines //Programming and Computer Software, Vol. 22, No. 2, 1996. pp. 68-74.
12. Райхлин В.А., Классен Р.К. Сравнительно недорогие гибридные технологии консервативных СУБД больших объемов //Информационные технологии и вычислительные системы. 2018. Т. 68. №1. С. 46-59.
13. Райхлин В.А., Минязев Р.Ш. Мультикластеризация распределенных СУБД консервативного типа //Нелинейный мир,2011. №8. С.473-481.
14. Классен Р.К. Особенности эффективной обработки SQL-запросов к базам данных консервативного типа

//Информационные технологии и вычислительные системы. 2018. Т.68.№4. С.108-118.
15. Oracle. The MySQL Plugin API //MySQL Documentation. 2018. URL: https://dev.mysql.com/doc/refman/5.7/en/plugin-api.html (дата обращения: 09.04.2018).
16. Райхлин В.А. Конструктивное моделирование систем. – Казань: Изд-во «Фэн» («Наука»), 2005. – 304 с.
17. Haken, Hermann. (2004). Synergetics: Introduction and Advanced Topics. 10.1007/978-3-662-10184-1.
18. Klassen R.K.: PerformSys. https://github.com/rozh1/PerformSys/ (2018). (дата обращения: 09.12.2018)
19. Martin J. Computer database organization. 2nd ed. New Jersey 07632: Prentice-Hall, Inc., Englewood Cliffs, 1977. 713 pp.
20. Raikhlin V.A., Klassen R.K. Can GPU-accelerator significantly increase the effectiveness of conservative DBMS considerable volumes on cluster platforms? //2017 International Siberian Conference on Control and Communications (SIBCON). 2017. DOI: 10.1109/SIBCON.2017.7998474
21. CoGaDB – Column-oriented GPU-accelerated DBMS. URL: http://cogadb.cs.tudortmund.de/wordpress. (дата обращения: 29.01.2019)
22. PGStrom 2016. URL: https://wiki.postgresql.org/index.php?title=PGStrom&oldid=25517. (дата обращения: 05.10.2018)
23. Rauhe H. Finding the Right Processor for the Job Co-Processors in a DBMS, Ilmenau University of Technology, Ilmenau, Dissertation urn:nbn:de:gbv:ilm1-2014000240, 2014.
24. Wenbin F., Bingsheng H., Qiong L. Database Compression on Graphics Processors //Proc. VLDB Endow., Vol. 3, No. 1-2, Sep 2010. P.670-680.
25. Bres S. Efficient query processing in co-processor-accelerated database. PhD dissertation, University of Magdeburg (2015).
26. Klassen R.K.: Clusterix-N. https://bitbucket.org/rozh/clusterixn/ (2019). (дата обращения: 10.03.2019)

Классен Роман Константинович. Казанский национальный исследовательский технический университет им. А.Н. Туполева, г. Казань. Старший преподаватель кафедры компьютерных систем. Окончил аспирантуру в Казанском национальном исследовательском техническом университете им. А.Н. Туполева – КАИ в 2019 году. Количество печатных работ: 14. Область научных интересов: высокопроизводительные системы, big data, информационные технологии. E-mail: klassen.rk@gmail.com

Райхлин Вадим Абрамович. Казанский национальный исследовательский технический университет им. А.Н. Туполева, г. Казань. Профессор кафедры компьютерных систем, доктор физ.-мат. наук, профессор. Окончил Казанский авиационный институт им. А. Н. Туполева (КАИ) в 1962 году. Количество печатных работ: 153 (в т.ч. 29 монографий и учебных пособий). Область научных интересов: конструктивное моделирование систем. E-mail: no-form@evm.kstu-kai.ru

Improving the Efficiency of Clusterix-Like DBMS for Big Data Analytical Processing

R. K. Klassen, V. A. Raikhlin

Kazan National Research Technical University named after A. N. Tupolev - KAI, Kazan. Russia

Abstract. Commercial OLAP-systems are economically unavailable for organizations with limited financial capabilities. Analytical processing large amounts of data in these organizations can be accomplished using open source software systems on a cost-effective cluster platform. Previously created Clusterix-like DBMS were not efficient enough according to the «performance/cost» criterion. With a view to the enhance the effectiveness of such systems in the article considers their further development with a focus on a full load of processor cores and the using GPU acceleration (systems Clusterix-N, N – from New) up to the development of a system comparable in efficiency to the open source system Spark, which is currently considered the most promising. The development methodology was based on the constructive system modeling methodology.

Keywords: analytic processing of significant data volumes, open source software systems on a cluster platform, increasing the efficiency of Clusterix-like DBMS, full loading of processor cores, full load of processor cores, GPU acceleration, comparison with Spark, accepted methodology.

DOI 10.14357/20718632190405

References

1. E. F. Codd. Providing OLAP to user-analysts: an it mandate, Apr. 1993. Technical Report, E. F. Codd and Associates.
2. Microsoft. Parallel Query Processing //Resources and Tools for IT Professionals | TechNet. 2018. URL: https://technet.microsoft.com/en-us/library/ms178065(v=sql.105).aspx (accessed: 05.04.2018).
3. Lenovo System x3950 X6 // TPC-H Result Highlights. 2016. URL: http://www.tpc.org/3321 (accessed: 10.08.2018).
4. Lenovo. System x3950 X6 Rack Server //Lenovo official website in Russia. 2017. URL: https://www3.lenovo.com/ru/ru/data-center/servers/mission-critical/System-x3950-X6/p/WMD00000002 (accessed: 15.07.2018).
5. Oracle Exadata Database Machine X7 //Oracle Russia and CIS. 2018. URL: https://www.oracle. com/ru/engineered-systems/exadata/database-machine-x7/index.html (accessed: 10.08.2018).
6. EMC Education Services. Data Science and Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data // John Wiley & Sons. 432 p.
7. Xin, Reynold & Rosen, Josh & Zaharia, Matei & J. Franklin, Michael & Shenker, Scott & Stoica, Ion. (2012). Shark: SQL and Rich Analytics at Scale. Proceedings of the ACM SIGMOD International Conference on Management of Data. 10.1145/2463676.2465288.
8. Russian DBMS industry advances on «elephants» [Rossijjskaja otrasl' SUBD prodvigaetsja na «slonakh»]//Connect. 2017. No. 5-6. pp.34-38.
9. Postgres Pro DBMS //Postgres Professional. 2018. URL: https://postgrespro.ru/products/postgrespro (accessed: 03.05.2018).
10. Hellerstein J.M., Stonebraker M., Hamilton J. Architecture of a Database System //Foundations and Trends in Databases. 2007. Vol. 1. No. 2. pp. 141-259.
11. Raikhlin V.A. Simulation of Distributed Database Machines //Programming and Computer Software, Vol. 22, No. 2, 1996. pp. 68-74.
12. Raikhlin V.A., Klassen R.K. Sravnitel'no nedorogie gibridnye tekhnologii konservativnykh SUBD bol'shikh ob"-emov [Relatively inexpensive hybrid technology of large volumes conservative DBMS] //Journal of Information Technologies and Computing Systems. 2018. Vol 68. №1. P. 46-59.
13. Raikhlin V.A., Minjazev R.Sh. Mul'tiklasterizaciya raspredelennyx SUBD konservativnogo tipa [Multiclusterization of distributed dbms of conservative type] // Nonlinear world, 2011. №8. P.473-481.
14. Klassen R.K. Osobennosti ehffektivnojj obrabotki SQL zaprosov k bazam dannykh konservativnogo tipa [Features of efficient processing of SQL-queries to conservative type databases] // Journal of Information Technologies and Computing Systems. 2018. Vol 68. №4. P. 108-118.
15. Oracle. The MySQL Plugin API //MySQL Documentation. 2018. URL: https://dev.mysql.com/doc/refman/5.7/en/plugin-api.html (accessed: 09.04.2018).
16. Raikhlin V.A. Konstruktivnoe modelirovanie sistem [Constructive system modeling]. – Kazan. Publisher: «Feng» («Nauka» [«Science»]), 2005. – 304 pp.
17. Haken, Hermann. (2004). Synergetics: Introduction and Advanced Topics. 10.1007/978-3-662-10184-1.
18. Klassen R.K.: PerformSys. https://github.com/rozh1/PerformSys/ (2018). (accessed: 09.12.2018).
19. Martin J. Computer database organization. 2nd ed. New Jersey 07632: Prentice-Hall, Inc., Englewood Cliffs, 1977. 713 pp.
20. Raikhlin V.A., Klassen R.K. Can GPU-accelerator significantly increase the effectiveness of conservative DBMS considerable volumes on cluster platforms? //2017 International Siberian Conference on Control and Communications (SIBCON). 2017. DOI: 10.1109/SIBCON.2017.7998474
21. CoGaDB – Column-oriented GPU-accelerated DBMS. URL: http://cogadb.cs.tudortmund.de/wordpress. (accessed: 29.01.2019)
22. PGStrom 2016. URL: https://wiki.postgresql.org/index.php?title=PGStrom&oldid=25517 (accessed: 05.10.2018).
23. Rauhe H. Finding the Right Processor for the Job Co-Processors in a DBMS, Ilmenau University of Technology, Ilmenau, Dissertation urn:nbn:de:gbv:ilm1-2014000240, 2014.
24. Wenbin F., Bingsheng H., Qiong L. Database Compression on Graphics Processors //Proc. VLDB Endow., Vol 3, No. 1-2, Sep 2010. P.670-680.
25. Bres S. Efficient query processing in co-processor-accelerated database. PhD dissertation, University of Magdeburg (2015)
26. Klassen R.K.: Clusterix-N. https://bitbucket.org/rozh/clusterixn/ (2019). (accessed: 10.03.2019).

Klassen Roman Konstantinovich. Department for Computer Systems, Institute for Computer Technologies and Information Protection, Kazan National Research Technical University named after A. N. Tupolev - KAI, Kazan. Russia, e-mail: klassen.rk@gmail.com

Raikhlin Vadim Abramovich. Department for Computer Systems, Institute for Computer Technologies and Information Protection, Kazan National Research Technical University named after A. N. Tupolev - KAI, Kazan. Russia, e-mail: no-form@evm.kstu-kai.ru