Сравнительно недорогие гибридные технологии консервативных СУБД больших объемов

В.А. Райхлин, Р.К. Классен

Аннотация. Обсуждаются вопросы организации консервативных СУБД на сравнительно недорогих кластерных платформах с применением средств MySQL[1] и GPU-акселераторов на исполнительном уровне. Актуальность принятой ориентации на работу с базами данных больших объемов определяется современными тенденциями интеллектуальной обработки больших информационных массивов. Повышение объема баз данных требует их хеширования по узлам кластера. Это обуславливает необходимость использования регулярного плана обработки запросов. Применение однородных кластерных технологий (СУБД Clusterix, оптимальная конфигурация «совмещенная симметрия») требует дополнительно динамической сегментации промежуточных и временных отношений. Для управления базами данных больших объемов предложены гибридные технологии (проекты Clusterix-N и Clusterix-G), что позволяет исключить динамическую сегментацию и значительно повысить эффективность по критерию производительность/стоимость. В них данные располагаются в оперативной памяти ОП узлов. Отличительной особенностью СУБД Clusterix-G с GPU-акселераторами является работа со сжатыми базами данных, что позволяет увеличить их объем при ограниченных объемах ОП. Предложенные технологии более эффективны в сравнении с Clusterix, а по производительности Clusterix-G должна в разы превышать Clusterix-N для интерконнекта среднего быстродействия.

Ключевые слова: консервативные СУБД, большие данные, доступные кластерные платформы, регулярный план обработки, гибридные технологии, сжатие баз данных, применение графических ускорителей, сравнительные оценки эффективности и производительности.

Введение

Наше рассмотрение ориентировано на построение высокоэффективных (по критерию производительность/стоимость) параллельных СУБД *консервативного типа* сравнительно больших объемов довольно скромными средствами, доступными в настоящее время широкому кругу научно-образовательных организаций. Под *консервативностью* авторами понимается эпизодическое обновление данных в специально выделяемое время, т.е. исключение из процесса обработки запросов в динамике функционирования системы операций *insert*, *delete* и *update*. Его актуальность определяется тенденциями *интеллектуальной обработки* информационных массивов. Примером тому является проект специализированной системы *SciDB* для обработки научных данных (результатов испытаний, наблюдений за состоянием среды и др.) [1-3]. Для них характерен высокий удельный вес именно сложных запросов типа *select – project – join*, оперирующих

[1] Используемые обозначения

Clusterix, Clusterix-M, Clusterix-N, Clusterix-G, PerformSys – названия затрагиваемых в статье проектов консервативных СУБД на кластерной платформе, принятые разработчиками этих проектов.

MySQL – применяемая в этих проектах инструментальная СУБД.

GPU (Graphics Processing Unit) – сопроцессор, широко применяемый как ускоритель трехмерной графики. Используется и при выполнении обычных вычислений – подход GPGPU (General-Purpose GPU).

CPU – центральный процессор.

TPC (Transaction Processing Performance Council) – Совет по производительности обработки транзакций.

TPC-H – один из тестов производительности систем принятия решений.

множеством таблиц с большим числом операций соединения *join*. При ограниченных объемах оперативной памяти узла, такие базы данных приходится хешировать по узлам кластера. Единственно приемлемым планом обработки запросов в таком случае оказывается регулярный [4] (Рис. 1) по схеме СЕЛЕКЦИЯ (σ) – ПРОЕКЦИЯ (π) – СОЕДИНЕНИЕ (σ_θ (R x S)).

Здесь < x > – декартово произведение, селектирование в операции соединения ведется по θ-соответствию кортежей отношений R и S. По условию соединение всегда естественное и проводится по полям первичного ключа. При использовании стратегии «*множество узлов кластера – на один запрос*» база данных оказывается распределенной по узлам. Получение любого промежуточного R_i' и любого временного $R_{B(i-2)}$ отношений происходит параллельно. При этом теоретически возможно совмещение обоих процессов, если за время съема с дисков и предварительной обработки (селекции с проекцией) исходного отношения R_i успевает сформироваться временное отношение $R_{B(i-2)}$.

Для целей исследования была разработана натурная модель – СУБД *Clusterix* [5]. Обработка запросов в ней – двухуровневая. На нижнем уровне выполняются операции селекции (σ) и проекции (π) над исходными отношениями R_i базы данных. Результатом обработки этого уровня является промежуточное отношение R_i'. На верхнем уровне реализуется операция соединения $R_{Bi-1} = \pi\ (\sigma_\theta\ (R_i' x R_{Bi-2}))$, где R_{Bi-1} и R_{Bi-2} – временные отношения как результаты соединений в i- и в (i-1)-шагах соответственно. Фильтрация на нижнем уровне значительно снижает объемы данных, передаваемых на верхний уровень.

Отношения БД распределены по дискам на процессорах нижнего уровня (IO_r) с применением хеш-функции к первичному ключу для каждого кортежа отношения. Требованию равномерного распределения кортежей хорошо удовлетворяет выбор в качестве нее функции деления по модулю [6]. Основание деления – число процессоров нижнего уровня. Остаток от деления r однозначно определяет номер страницы (процессора IO_r), куда будет распределен кортеж. Процессоры верхнего уровня обработки запроса называются процессорами JOIN.

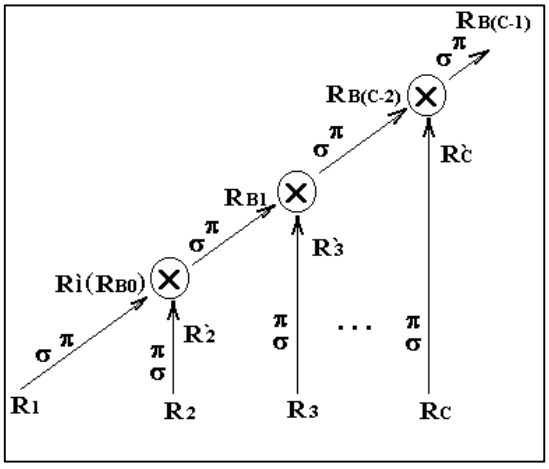

Рис. 1. Регулярный план

Кроме исполнительных процессоров (IO_r и $JOIN_j$), есть еще два процессора. Процессор УПР реализует функции мониторинга и управления остальными процессорами системы. Для реализации функций объединения результатов обработки с уровня JOIN введен процессор SORT. Он дополнительно выполняет функции агрегации (SUM(), AVG(), MAX(), MIN() и др.) и сортировки результата предшествующей операции соединения. В *Clusterix* реализован вариант совместной работы процессоров УПР и SORT на Host ЭВМ. Работа на уровне файловой системы, системных буферов, алгоритмов доступа к данным, работы с индексами и т.п. реализуется с помощью стандартной СУБД MySQL.

Параллельная СУБД Clusterix представляет композицию четырех программных модулей: *mlisten, msort, irun, jrun*. Модуль *mlisten* реализует функции логических процессоров УПР и ПРТ, модуль *msort* – SORT, модуль *irun* – IO, модуль *jrun* – JOIN.

Модуль *mlisten* функционирует на отдельном узле (так называемой Host-машине). Реализует функции процессора УПР и претранслятора ПТР. Основными функциями этого модуля являются:

– получение запроса от пользователя и передача результатов обработки обратно;

– преобразование входного запроса в дерево обработки;

– пересылка соответствующих пакетов SQL-команд модулям *irun, jrun, msort*;

– мониторинг работы системы;

– сбор статистической информации.

Модуль *msort* предназначен для выполнения операции объединения результатов обработки JOIN с последующим выполнением операций агрегации и сортировки результата. Данный модуль функционирует на Host-машине.

Модуль *irun* реализует функции IO-уровня обработки данных – выполнение операций селекции и проекции над исходными отношениями базы данных.

Модуль *jrun* реализует функции JOIN-уровня обработки – выполнение операции соединения над промежуточным отношением, которое приходит от модуля *irun*, и над временным отношением, которое формируется в результате предыдущего соединения.

Загрузочные модули для обработки на верхних и нижних уровнях формируются из исходного SQL-запроса претранслятором. Задачей разработанного претранслятора является преобразование входного запроса пользователя в команды плана обработки рис.1 в виде трех пакетов SQL-команд (для IO, для JOIN и для SORT) с динамическим созданием индексов при выполнении операции соединения.

Запросы пользователей соответствуют следующему шаблону SQL-запроса:

SELECT список_атрибутов FROM список_отношений [WHERE условие_1[[AND условие_k] ...]] [GROUP BY список_атрибутов] [ORDER BY список_атрибутов].

Претрансляция проводится с учетом того, что:

– Предикаты запроса вида:

<имя_отношения.имя_атрибута><условие><константа>

перемещаются на уровень IO. Тем самым реализуется фильтрация данных на более ранних стадиях обработки. Это позволяет уменьшить объем данных при их маршрутизации по сети.

– Предикаты запроса вида:

<отношение_A.атрибут_B><условие><отношение_A. атрибут_C >

как условия селекции перемещаются на уровень IO.

– При выполнении операции проекции из исходного отношения выбираются только те атрибуты, которые непосредственно участвуют в обработке. Это условие касается как уровня IO, так и уровня JOIN, что обеспечивает максимальную фильтрацию данных при их продвижении по дереву обработки запроса.

– У временных и промежуточных отношений автоматически создается индекс по атрибуту, участвующему в операции соединения.

– В системе реализованы алгоритмы с использованием хеширования. Поэтому обрабатываются только запросы с эквисоединением.

При экспериментальном исследовании эффективности системы претранслятор не использовался. Запросы размещались в специальном интерфейсном отношении *query* системной базы данных Clusterix в оттранслированном и готовом для исполнения виде. Пакеты команд пользовательского запроса считывались из этого отношения и передавались соответствующим модулям для исполнения.

Система реализует потоково-конвейерный механизм обработки запросов. Конвейерность достигается за счет совмещения работ на нижнем, верхнем уровнях обработки и на уровне SORT. Для обеспечения надежной работы системы на всех уровнях используется барьерная синхронизация. Применение динамической сегментации промежуточных и временных отношений позволяет реализовать параллельное выполнение операции соединения на множестве процессоров $JOIN_j$. Вместе с тем, это приводит к параболической зависимости времени T обработки представительского теста ПТ от числа узлов кластера h из-за увеличения нагрузки по интерконнекту с ростом h.

На Рис. 2 показан результат эксперимента [7] на натурной модели СУБД *Clustrerix*, полученный для кластерной платформы: вычислительный кластер фирмы SUN из 22 узлов 2 Quad-core Intel Xeon E5450CPU/1,87GHz/32GB. Интерконнект между узлами – Gigabit Рис.2. $V_{БД}$=5GB: монокластер, «линейка», ПТ Ethernetc 24-портовым коммутатором Cisco. Дисковая подсистема узла – SAS диски XRB-SS2CD 146G10KZ с пропускной способностью 300 MB/s. На каждый компьютер установлена операционная система Linux. Все процессоры дополнительно оснащены СУБД MySQL. В одном узле совмещено функционирование логических процессоров IO и Join (конфигурация «линейка»). В качестве ПТ использовался ограниченный тест TPC-H из 14 запросов, не содержащих операций записи. Граница масштабируемости по числу узлов h_G отвечает точке

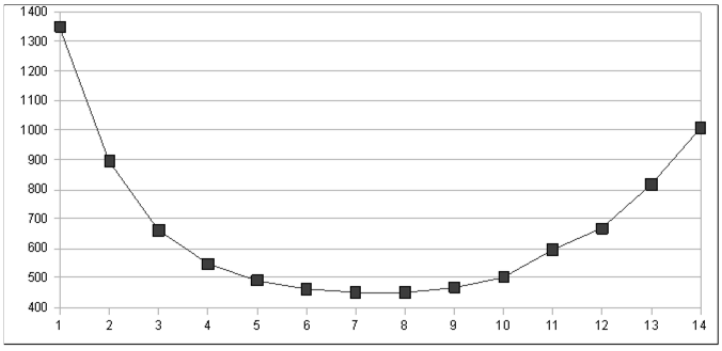

Рис. 2. $V_{БД}$=5GB: монокластер, «линейка», ПТ

минимума графика T = f(h). Для Рис. 2 значение h_G= 7. Наличие двух процессоров в узле позволяет существенно улучшить масштабируемость «линейки» и повысить ее максимальное быстродействие на границе масштабируемости. Это достигается установкой на каждый SMP-узел вычислительного кластера двух серверов MySQL. Один из них связан с процессами IO, другой – с процессами Join (конфигурация «совмещенная симметрия»).

Переход к мультикластеризации (СУБД *Clusterix-M* [7]) означает изменение идеологии обработки запросов в кластерной системе: от «множества маломощных узлов на один запрос» к одному мощному узлу на каждый запрос». При этом «мощный узел» реализуется как монокластер – компонент кластера в целом с установкой на него системы *Clusterix*. Число узлов монокластера не превышает грани масштабируемости.

Host (главная ЭВМ) включает модуль ROUTER, который распределяет поток запросов, поступающих от N клиентов между n монокластерами-компонентами. Очередной запрос от каждого пользователя поступает на вход сервера СУБД сразу, как будет получен ответ на предыдущий запрос. В любой момент времени суммарное число запросов, находящихся в сервере, $L = N \geq n \cdot r$. Оптимальная длина очереди монокластера r = 2.

На Рис. 3 показаны результаты обработки на прежней платформе конкатенации трех запросов ПТ с объемом базы данных 5GB. Монокластеры-компоненты конфигурировались как «совмещенная симметрия». Варьировалось количество монокластеров n и число их узлов h_m. В случае h_m=4 наблюдался рост масштабируемости мультикластера по сравне нию с однокластерной системой в 2,3 раза; при h_M=6 – более чем в 2,6 раз. Степень роста производительности на пороге масштабируемости – в 2,4 и не менее чем в 3 раза соответственно.

Таким образом, при использовании обычных вычислительных кластеров, организация обработки запросов к консервативным БД больших объемов по регулярному плану – пока что объективная закономерность. Это обусловлено необходимостью хеширования таких БД на множестве узлов кластера. Столь же обосновано и динамическое сегментирование промежуточных и временных отношений. Следствием

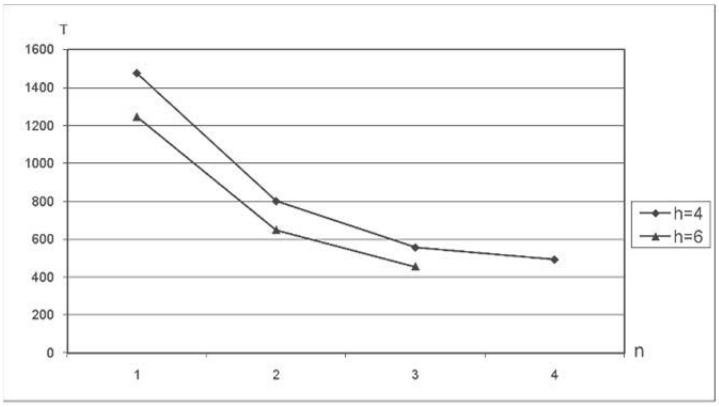

Рис. 3. $V_{БД}$=5GB: мультикластер, «совмещенная симметрия», конкатенация 3-х перестановок ПТ

указанных обстоятельств оказывается «квазипараболическая» зависимость времени обработки ПТ от числа узлов (для вычислительных кластеров более характерно приближение к гиперболе). При этом максимум производительности на грани масштабируемости оказывается достаточно низким. Такова в данном случае плата за работу с базами данных больших объемов. Переход к мультикластеризации при ограничении числа узлов в монокластерах-компонентах значительно улучшает положение, приближая указанную зависимость к гиперболе. Но все же и здесь достижимая производительность оставляет желать лучшего.

В статье показывается, что исправить положение можно *принципиальным изменением архитектуры Clusterix* путем отказа от принципа «однородности» (характерного для конфигурации «совмещенная симметрия») в пользу «гибридности», подразумевающей разделение кластера на две различные части – блоки IO и JOIN. Это позволяет исключить распределенную обработку запроса на уровне JOIN и динамическое сегментирование промежуточных и временных отношений, что существенно облегчает разработку СУБД в целом и повышает ее эффективность (СУБД *Clusterix-N*, N – от New). Дальнейшего улучшения можно достигнуть подключением к каждому узлу кластера графических ускорителей – GPU-акселераторов (СУБД *Clusterix-G*, G – от GPU).

1. Формулировка задачи

Чтобы обеспечить возможность сравнения с результатами, полученными для СУБД *Clusterix-M* (Рис. 3), рассмотрение ограничивается случаем равенства числа узлов IO и Join на примере $n_{IO} = n_{Join} = 3$ (рис. 4, рассматриваемый пример G- системы, узлы (1-3) – IO, узлы (4-6) – Join. Исключив GPU, имеем N-систему), $V_{БД} = 5$ GB, конкатенации трех ПТ. Такая конфигурация отвечает принятым *к будущей разработке натурным моделям Clusterix-G и Clusterix-N*, которые будут организованы на платформе GPU-кластера КНИТУ-КАИ с шестью исполнительными узлами (узлы 1-6) и одним управляющим (Mgm). Параметры узлов: 2 six-core E5-2640CPU/ 2,5GHz/DDR3 128GB; 2 448-core GPU Tesla C-2075/ 1,15GHz/GDDR5 6GB (на Mgm GPU отсутствуют). Дисковая подсистема узла – RAID-массив 4 WD1000 DHTZ/1TB. Операционная система – SUSE Linux Enterprise Server версии 11. Интерконнект между узлами – GigabitEthernet с 24-портовым коммутатором SSE G24-TG4.

Как и ранее, база данных хешируется на уровне узлов IO. По условию суммарный объем оперативной памяти узлов IO допускает размещение в нем всей базы данных. На уровне узлов Join используется стратегия «запрос на ядро» (реализуемость такой стратегии средствами MySQL доказана в работе [8]). Соединение выполняется в виде единой процедуры

$$R_1' \mathit{join} (\mathit{join}\ R_2'(\mathit{join}\ R_3'(\ldots))) \ldots)$$

по каждому запросу ПТ (ее последовательное выполнение согласно регулярному плану рис.1 оказывается вдвое медленнее). Для улучшения быстродействия систем все проекции «опущены вниз» (на уровень IO). В СУБД *Clusterix-N* стратегия «запрос на ядро» используется и на уровне узлов IO. В *Clusterix-G* стратегия на этом уровне – «узел на запрос».

Графические ускорители (GPU) уже применяются для ускорения работы СУБД CoGaDb [9], как расширение PG-Storm для Postgres [10] и др. Использование GPU позволяет существенно снизить время выполнения отдельных операций. При объеме базы данных $V_{БД} \leq 3$GB вся БД может храниться в глобальной памяти GPU (6GB для GPU Fermi). Эффективность такой СУБД будет достаточно высока. При значительных $V_{БД}$ необходимость обмена данными с GPU существенно снижает производительность. 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 [11], в то время как для шины PCI-e 2.0x16 – 6,4 GB/s [12]. Именно по этой причине достигнутый в [13] рост производительности сервера БД от использования GPU не превысил 40%. При использовании кластерных платформ интерконнект должен оказывать большее влияние на производительность системы, чем шина PCI-e.

На наш взгляд, целесообразность подключения к узлам кластера GPU-акселераторов диктуется, прежде всего, необходимостью увели-

чения объемов обрабатываемых баз данных. Для построения СУБД *Clusterix-G* предлагается хранить распределенную базу данных в оперативной памяти IO-узлов в поблочно-сжатом виде, передавать по сети сжатые порции отношений R_i' и возможно максимально загружать ядра многоядерных процессоров на уровне JOIN с применением в них GPU-акселерации для разжатия поступающих данных. Размеры блоков таковы, что после проецирования и разжатия они гарантированно умещаются в рабочей памяти GPU. Оптимальный коэффициент сжатия данных по алгоритму RLE (Run Length Encoding) [14], обеспечивающий минимизацию суммарного времени передачи и разжатия данных в GPU, равен 5 [15]. Этот минимум составляет 60% времени передачи несжатых данных того же объема.

Алгоритм подготовки данных для сжатия следующий (реализуется при формировании сжатой базы данных):

1. Найти самое «длинную» запись (длиной RS) в обрабатываемом отношении R (в общем случае некоторые отношения могут содержать поля переменной длины).

2. Найти количество записей RC в R, 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 (лучшие показатели по сжатию имеют СУБД, ориентированные на хранение данных по столбцам [16]).

Для целей разработки натурной модели *Clusterix-G* принят следующий ориентировочный план ее функционирования. Процессор УПР в Mgm ведет очередь запросов, раздает задание на обработку, балансирует нагрузку между узлами IO (в отличие от *Clusterix* и *Clusterix-M*, они работают асинхронно), получает промежуточные и конечные результаты исполнения запросов. Процессор SORT выполняет функции агрегации и сортировки результата обработки запроса. Процессор ROUTER – функцию балансировки нагрузки между узлами JOIN. В буфере BUF по каждому запросу осуществляется конкатенация частей сжатых отношений R_i', подлежащих дальнейшему соединению. Претрансляция запросов ПТ к регулярному плану не проводится. Загрузочные модули IO и JOIN этих запросов хранятся в Mgm.

CPU в IO проецируют колонки, управляют очередью своих команд, работой графических ускорителей (2 GPU на узел), пересылкой сжатых данных по шине PCI-e (связывающей CPU и GPU) и сети GigabitEthernet (связывающей IO и Mgm). GPU в IO выполняют функции разжатия данных, их селектирования и сжатия полученных результатов. На уровне JOIN реализуются интегрированные процедуры *«join»* по запросу в целом. Средствами MySQL 5.7 оптимизируется их выполнение. Все отношения R_i' по каждому запросу должны поступить в память узла JOIN перед началом процедуры. GPU в узлах JOIN реализуют функцию разжатия блоков сжатых данных, поступающих в эти узлы, «на проходе».

Основной задачей статьи является получение теоретических оценок времени выполнения тестового пакета для СУБД *Clusterix-N и Clusterix-G* при оговоренных ранее условиях. При этом будем полагать, что этапы обработки теста в целом (*«select–project»*-обработка всего теста → передача ее результата с уровня IO к Mgm и далее – от Mgm на уровень JOIN → собственно *«join»*-обработка) следуют друг за другом. При выполнении расчетов будем учитывать дуплексный характер шины PCI-e, который позволяет параллельную загрузку-выгрузку данных и параллельное селектирование в двух GPU.

Аналогичный дуплекс в сети GigabitEthernet позволит на практике частично совместить передачи от IO к Mgm и от Mgm к JOIN, плюс частичные совмещения приема данных и обработки в JOIN. По условию принятое последовательное выполнение по этапам обработки должно компенсировать реальные затраты времени на функционирование Mgm и реализацию управления в кластере.

Дополнительно будем учитывать следующее [15]:

1. На множестве 14 запросов ПТ для запроса №9 суммарное время *«select–project»*-обработки и время выполнения единой операции *«join»* максимальны. При отсутствии GPU на одноядерном процессоре с частотой 2,5 GHz

они составляют соответственно $11{,}4V_{БД}$ и $14{,}5V_{БД}$ секунд, где $V_{БД}$ – в GB.

2. Суммарный объем отношений R_i, необходимый для выполнения ПТ $V_{\Sigma}^{Ri} \approx 9{,}7\ V_{БД}$ GB. Суммарный объем промежуточных отношений R_i' для ПТ $V_{\Sigma}^{Ri'} \approx 1{,}4V_{БД}$ GB.

3. При действии ПТ суммарное время передачи сжатых отношений R_i' от Mgm на уровень JOIN определяется с поправкой на отсутствие передач R_i' для запросов №1 и №6, не содержащих операций *«join»*. Сканкатенированные в BUF Mgm части R_i' этих запросов будут ретранслированы Mgm без дополнительной *«join»*-обработки. Сумммарный объем R_i' по этим запросам примерно равен $0{,}14V_{БД}$ GB. После сжатия имеем $0{,}028V_{БД}$ GB. Так что в блок JOIN поступает $(V_{\Sigma}^{Ri'})_{Join} \approx 1{,}37V_{БД}$ GB.

2. Сравнительные теоретические оценки

По Clusterix-N. Случай $V_{БД}$ = 5GB, конкатенация 3 ПТ; 3 IO, хеширование БД, 3 ПТ обрабатываются последовательно, один за другим; 3 Join, по 1 ПТ на каждый. Времена параллельного функционирования блоков IO и Join определятся соответственно величинами $t_{\Sigma}^{\sigma,\pi}$ и $(t_{\Sigma}^{join})^{12}$ для самых «тяжелых» на ПТ запросов. В обоих случаях таковым является запрос №9. Для него

$t_{\Sigma}^{\sigma,\pi} \approx 11{,}4 \cdot 5 \cdot 3/3 = 57$ с.

Время передачи по сети GigabitEthernet пакета объемом $V_{\Sigma}^{Ri'}$ составит

$t_n'' \approx 1{,}4 \cdot 5 \cdot 3/0{,}1 = 210$с.

Так что суммарное время *«select–project»*-обработки

$(t_{\Sigma}^{\sigma,\pi})' = t_{\Sigma}^{\sigma,\pi} + t_n'' \approx 267$ с.

Значение

$(t_{\Sigma}^{join})^{12} \approx 14{,}5 \cdot 5 = 72{,}5$ с.

К этому следует плюсовать время $t_{уд}$ на удаление средствами MySQL поступивших в узел JOIN промежуточных отношений. Как показал предварительный эксперимент, это время не превышает 20% времени выполнения единых операций *«join»* для всех запросов, если удаление проводить по окончании обработки каждого запроса.

$t_{уд} \approx 0{,}2(t_{\Sigma}^{join})^{12} = 14{,}5$ с.

Время передачи по сети GigabitEthernet от Mgm к JOIN пакета объемом $(V_{\Sigma}^{Ri'})_{Join}$

$t_n''' \approx 1{,}37 \cdot 5 \cdot 3/(0{,}1$ GB/s) = 189 с.

Общее время *«join»*-обработки

$(t_{\Sigma}^{join})' = (t_{\Sigma}^{join})^{12} + t_{уд} + t_n''' \approx 72{,}5 + 14{,}5 + 189 = 276$с.

Суммарное время обработки теста в целом

$T = (t_{\Sigma}^{\sigma,\pi})' + (t_{\Sigma}^{join})' \approx 267 + 276 = 543$ с.

Сравнение с данными рис.3 показывает, что применение рассмотренного подхода на 7 узлах должно позволить в данном случае получить то же быстродействие, что и на 12 узлах мультикластера. Это – существенное улучшение.

По Clusterix-G. Условия прежние. Теоретическая скорость передачи по шине PCI-e равна 6,4GB/s. Получение на практике 3,2GB/s можно гарантировать. Поэтому для несжатой БД суммарное время передач информации объемом V_{Σ}^{Ri} в каждом IO-узле от CPU параллельно к двум GPU составит

$t_{n1} = 9{,}7 \cdot 5 \cdot 3/(3 \cdot 2 \cdot 3{,}2) \approx 7{,}58$ с,

а от GPU к CPU –

$t_{n2} = 1{,}4 \cdot 5 \cdot 3/(3 \cdot 2 \cdot 3{,}2) \approx 1{,}1$ с.

Хранение исходной БД в сжатом виде с коэффициентом сжатия K=5 уменьшает t_{n1} примерно на 40%. Соответственно для 14 запросов ПТ получаем

$t_n' = t_{n1}' + t_{n2}' \approx 7{,}58 \cdot 0{,}6 + (1{,}1/5) \approx 4{,}77$ с.

Дополнительно необходимо учесть время на передачу сжатых данных по сети GigabitEthernet от IO к Mgm

$t_n'' = 1{,}4 \cdot 5 \cdot 3/(5 \cdot 0{,}1) = 42$ с.

GPU селектирует ПТ последовательно запрос за запросом. Суммарное время такого селектирования одним ядром CPU примерно равно $91{,}4 \cdot 5 = 457$ с [15]. Полагая (как это принято считать) десятикратным ускорение от использования одного GPU, для двух GPU получаем

$t_{\Sigma}^{\sigma,\pi} \approx 457/20 = 22{,}85$ с.

Итоговое время *«select–project»*-обработки

$(t_{\Sigma}^{\sigma,\pi})' = t_n' + t_{\Sigma}^{\sigma,\pi} + t_n'' \approx 70$ с.

Время передачи сжатых данных по сети GigabitEthernet от Mgm к узлам JOIN

$t_n''' \approx 1{,}37 \cdot 5 \cdot 3/(5 \cdot 0{,}1) = 41{,}1$ с.

Времена собственно *«join»*-обработки $(t_{\Sigma}^{join})^{12}$ и удаления $t_{уд}$ остаются прежними (как и в *Clusterix-N*). По условию в Join-узлах передача принятых сжатых данных в GPU и их разжатие в ускорителях выполняются «на проходе». Тогда параллельная передача разжатых данных из двух GPU в CPU вносит дополнительную несущественную задержку, равную

$t_n'''' = 1{,}37 \cdot 5/(2 \cdot 3{,}2$GB/s) $\approx 1{,}1$ с.

Общее время *«join»*-обработки
$(t_{\Sigma}^{join})' = (t_{\Sigma}^{join})^{12} + t_{уд} + t_n''' + t_n'''' \approx 72{,}5 + 14{,}6 + 41{,}1 + 1{,}1 = 129{,}3$ с.

Суммарное время обработки теста в целом
$T = (t_{\Sigma}^{\sigma,\pi})' + (t_{\Sigma}^{join})' \approx 70 + 130 = 200$ с.

Таким образом, выигрыш в производительности по сравнению с СУБД *Clusterix-N* – примерно в 2,7 раз, т.е. на 63%. Это более чем в 1,5 раза превышает достигнутый в [13] эффект от использования графических акселераторов. Мы ассоциируем понятие производительности со временем обработки. Выигрыш в производительности в 2,7 раз трактуется нами как величина отношения $T_{обр1}/T_{обр2}=2{,}7$. Поэтому $T_{обр2} = T_{обр1}/2{,}7 = 0{,}37\ T_{обр1}$, т.е. на 63% меньше (рост производительности – на 63%). Именно такова трактовка производительности в работе [13], в которой достигнутый рост производительности GPU-сервера БД составил 40%. Этот результат мы стремимся улучшить.

Установленный выигрыш не должен зависеть от объема базы данных, ибо по условию все компоненты времени обработки теста, как для *Clusterix-N*, так и для *Clusterix-G*, прямо пропорциональны $V_{БД}$. Значения этих компонент при работе с базами данных повышенных объемов могут быть весьма значительными, особенно для интерконнекта.

3. Экспериментальная проверка правомерности итоговой оценки для Clusterix-N

Чтобы убедиться в правомерности итоговых временных оценок для *Clusterix-N*, полученных теоретически, был разработан демонстрационный прототип. Его тестирование проводилось на описанном в разделе 2 GPU-кластере КНИТУ-КАИ согласно структуре Рис. 4, но с использованием двух узлов IO. Тест и $V_{БД}$ – прежние. В качестве инструментальной СУБД использовался MySQL 5.7 в специальной конфигурации. Для IO конфигурация была подобрана с целью оптимизации выполнения операций *«select–project»*, для JOIN – для оптимизации работы с памятью и выполнения операций *«join»*. После запуска MySQL вся БД автоматически загружается с дисков в оперативную память IO.

Началом работы процессора УПР в составе MGM является поступление нового номера запроса из перечня претранслированных запросов. Первоначально статус запроса устанавливается в ожидание. Как только УПР обнаруживает, что все узлы IO (они реализуют принцип «отношение на ядро»), способны принять очередной подзапрос *select*, он отправляет его. Операция отправки запроса повторяется до тех пор, пока не будут отосланы все подзапросы на выборку для каждого из отношений или пока не будут загружены все ядра IO. Если все подзапросы для всех отношений запроса отправлены, а узлы IO загружены не полностью (есть свободные ядра), то отправляются подзапросы следующего запроса. Результат работы узлов IO поступает в BUF MGM в виде блоков данных. Блоки данных представляют собой набор строк из результата выполнения подзапроса *select*. Размер блоков устанавливается в настройках. Блоки накапливаются в буфере MGM по отношениям. Когда все блоки одного из отношений получены, оно отправляется в узел JOIN, который используется для выполнения запроса с данным отношением.

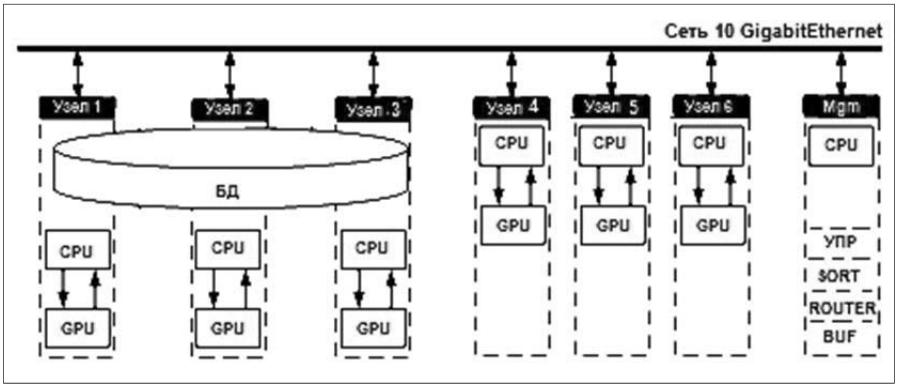

Рис. 4. Иллюстрация симметрии по числу узлов IO и Join

Балансировка нагрузки между узлами и ядрами JOIN осуществлялась по алгоритму *round-robin* (круговой алгоритм). При этом сначала в каждый JOIN загружается по одному запросу. Затем «по кругу» загружаются незанятые ядра всех узлов. Если заняты все ядра данного JOIN, переходим к следующему. После выбора JOIN с ним связывается тот запрос, для которого полностью получено хотя бы одно отношение, и начинается его передача. Аналогично происходит работа с процессором SORT: результат выполнения подзапроса *join* поступает буфер MGM и отправляется для дальнейшей обработки в SORT.

MGM содержит 14 претранслированных запросов без операций записи из состава теста TPC-H. Загрузочные модули *«select–project»*- и *«join»*-процедур поступают соответственно в узлы IO и JOIN, где без всяких преобразований передаются в MySQL, который их выполняет. Значительное время при выполнении *«join»*-процедур тратится на загрузку промежуточных отношений в MySQL и их индексацию, т.к. схемы баз данных в JOIN по каждому запросу уникальны.

IO взаимодействует с СУБД MySQL. БД распределяется между всеми узлами, на которых установлен данный модуль. Обработка подзапроса *select* осуществляется следующим образом. Исходный запрос модифицируется для возможности реализации чтения данных по блокам. К подзапросу добавляется строка вида: «LIMIT N,P», где N – размер блока данных, P – номер очередного блока. MySQL, выполняя модифицированный подзапрос *select*, возвращает строк не более, чем указано в настройках модуля. Результат работы MySQL передается в узел IO, где подвергается преобразованию и асинхронно передается по сети в BUF MGM. Выборка и передача данных по блокам позволяет избежать переполнения ОЗУ на узлах IO и загрузить сеть более равномерно.

Узел JOIN, как и узел IO, использует в качестве инструментальной СУБД MySQL. Он принимает задание и данные от УПР MGM, производит загрузку данных в MySQL и запускает непосредственно операцию *join*. Загрузка данных производится командой LOAD FILE, которая позволяет загружать данные из текстовых файлов. Подготовка загрузки включает: создание нового отношения в СУБД, создание первичных ключей и указание индексных полей. Результат *join*-обработки размещается в новом отношении, которое передается по готовности в MGM.

Процессор SORT использует свой MySQL. Как и узел JOIN, он получает задание и данные от УПР MGM, производит загрузку данных в MySQL и запускает непосредственно операцию *sort*. Результат работы процессора SORT размещается в новом отношении, которое передается по готовности в BUF MGM и от него – пользователю.

Демонстрационный эксперимент повторялся 5 раз на неизменном тесте, который представлял собой конкатенации 3-х первых рекомендуемых TPC-H перестановок ПТ. Среднее время выполнения теста после пяти запусков составило 506 с (максимальное – 560 с, минимальное – 430 с). На Рис. 5 показаны временные диаграммы, построенные для эксперимента со временем обработки теста T=515 с.

Таким образом, имеем хорошее соответствие теоретически полученной оценки со временем обработки теста на демонстрационном прототипе СУБД *Clusterix-N* при $V_{БД}$ = 5 GB. Это позволяет нам с высокой достоверностью говорить о реальности существенного повышения эффективности параллельных СУБД консервативного типа при последовательных переходах от архитектуры *Clusterix-M* к архитектуре *Clusterix-N* (с отказом от динамической сегментации промежуточных и временных отношений) при использовании сети GigabitEthernet.

4. Возможные проблемы и пути их преодоления

Реальная динамика процессов в системе при действии случайного потока запросов будет существенно более сложной по сравнению с рассмотренной. Ей свойственны серьезные «подводные камни». Один из них связан с тем, что объемы отношений, передаваемых по сети GigabitEthernet могут существенно различаться. Поэтому, чтобы совместить передачи от IO к Mgm и от Mgm к JOIN, эти передачи следует выполнять пакетами сравнительно небольших объемов.

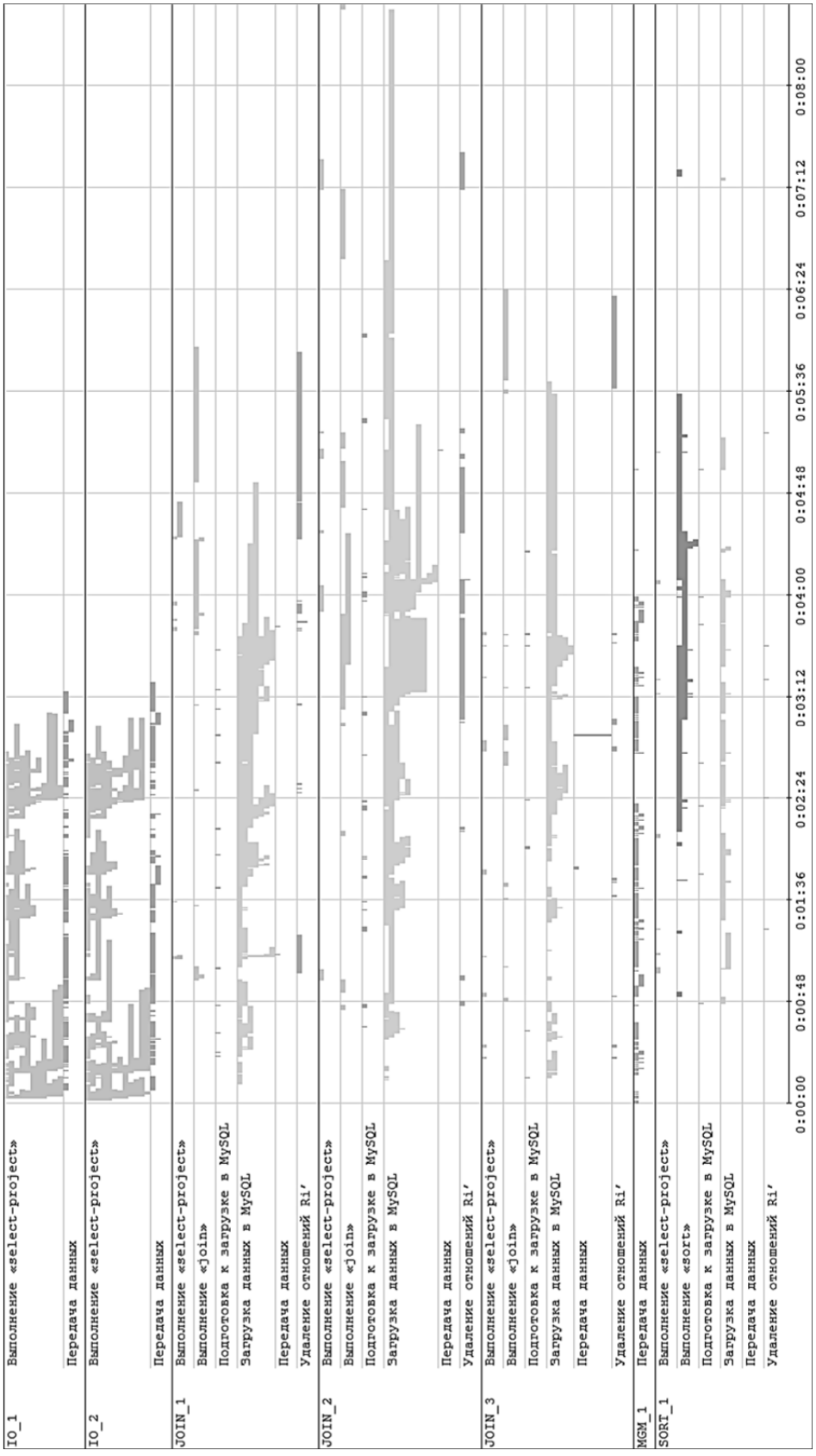

Рис.5. Экспериментальные временные диаграммы выполнения теста, СУБД *Clusterix-N*, $V_{БД}$ = 5 GB

Собственно *«join»*-обработка проводится с несжатыми данными. Их общий объем, например, для запроса №4 составляет 0,43 $V_{БД}$ GB. Поэтому для обеспечения работ с $V_{БД}$ = 1 TB (при условии реазизации единой процедуры *join* по запросу в целом) потребуется оперативная память объемом 512 GB (что современные серверы общего применения допускают) с переходом к асимметричным структурам, когда число узлов Join превышает число узлов IO (не все ядра узлов Join загружены).

Еще одним «подводным камнем» для СУБД *Clusterix-G* является возможная перегрузка BUF Mgm при действии непрерывного потока запросов, ибо суммарные времена *«select–project»*- и *«join»*-обработок существенно различаются. Этой опасности можно будет избежать введением со стороны Mgm запретов (при возникновении перегрузки) на передачу данных от IO. Неизбежные при таком подходе простои блока IO чреваты снижением производительности, но вполне оправданы. Рост числа узлов JOIN поможет разгрузить BUF Mgm от накопления R_i' по ожидающим исполнения запросам и будет способствовать повышению производительности системы.

Кроме того при $V_{БД} > 100$ GB целесообразно на уровне IO ввести барьерную синхронизацию (отказаться от асинхронной обработки), последовательно обрабатывая по одному запросу. Причина в том, что в противном случае BUF MGM может заполниться частичной информацией по множеству запросов. И ни один из них не будет передан в блок JOIN, ибо для реализации такой передачи надо сначала определить суммарный объем R_i' по обрабатываемому запросу и сравнить его с свободным объемом оперативной памяти каждого JOIN-узла

Найденная теоретическая оценка превышения быстродействия *Clusterix-G* над *Clusterix-N* справедлива только для сравнительно медленной сети GigabitEthernet. Нетрудно подсчитать, что использование сети 10GigabitEthernet повысит быстродействие СУБД *Clusterix-N* примерно в 3 раза, а СУБД *Clusterix-G* – всего в 1,6 раз. Так что выигрыш по производительности при переходе от *Clusterix-N* к *Clusterix-G* снизится до 33%. А при использовании сети Infiniband станет еще менее значительным. Так что основным эффектом от сжатия баз данных с использованием графических ускорителей следует считать снижение потерь времени на интерконнект и аппаратных затрат на реализацию блока IO.

Большинство организаций ориентировано на использование сравнительно недорогих вычислительных кластеров с объемом ОЗУ в узле не более 512 ГБ, что ограничивает допустимые размеры $V_{БД}$ – не более 1TB. Но если $V_{БД}$ не превышает 450 GB, то можно по иному (в сравнении с *Clusterix-N, G*) подойти к решению проблемы. Самостоятельное применение MySQL последних версий позволяет полностью использовать ресурсы многоядерных узлов (стратегия «ядро на запрос»). В работе [8] удалось обеспечить 100% загрузку всех ядер при специальной настройке MySQL 5.6 (технология *PerformSys*, по терминологии [17]). Полученный для СУБД *PerformSys* на платформе Sun-кластера при новых условиях эксперимента (ПТ – случайная последовательность из 800 запросов на множестве запросов ограниченного теста TPC-H, $V_{БД}$ = 5 GB) и хранении БД в оперативной памяти узла график зависимости среднего времени T выполнения ПТ в *минутах* от количества узлов в системе показан на Рис. 6. Сравнение с данными Рис. 2-Рис. 5 говорит о перспективности применения технологии *PerformSys* организациями с ограниченными финансовыми возможностями при $V_{БД} \leq 450$ GB.

Заключение

В статье предложены 2 технологии построения сравнительно недорогих консервативных СУБД повышенных объемов – *Clusterix-N* и *Clusterix-G*, названные гибридными по архитектурному признаку (в противовес однородным), с учетом возможного применения GPU-ускорителей и сжатия данных. Получены сравнительные оценки времени выполнения тестового пакета для этих СУБД при оговоренных условиях. Показано хорошее совпадение теоретических и экспериментальных оценок для *Clusterix-N*, что позволяет надеяться на аналогичное совпадение и для *Clusterix-G*. Установлено, что более эффективной альтернативой этим СУБД при умеренных объемах баз данных является СУБД *PerformSys*.

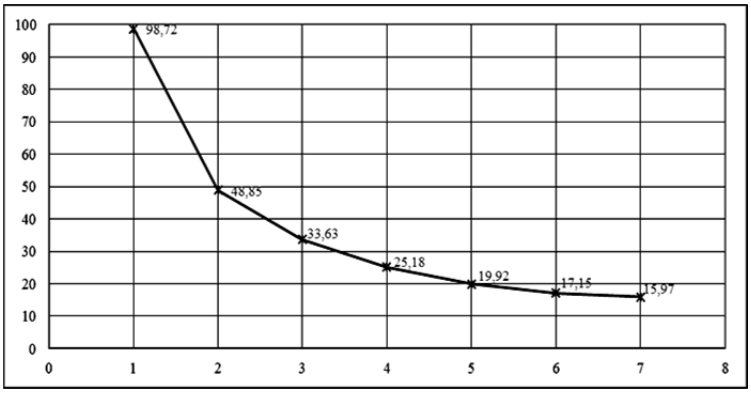

Рис.6. СУБД *PerformSys*, Sun-кластер $V_{БД}$=5GB: График T = f(n)

Ожидаемые результаты от применения рассмотренных технологий несопоставимы с достижимыми объемами обрабатываемых данных и производительностью современных дорогостоящих аппаратно-программных средств. Одним из них является система Lenovo x3950X6 (8 вычислительных модулей, 144 ядра) с внешней флеш-памятью ~120 ТВ и объемом оперативной памяти 12 ТВ [18]. Совокупная стоимость системы $2 634 342. При этом стоимость аппаратуры составляет ~$1,5 млн., а стоимость ПО (Windows Server 2016 и SQL Server 2016 Enterprise Edition) ~$1 млн. Далеко не все организации в состоянии приобрести столь дорогостоящую систему. Так, GPU кластер КНИТУ-КАИ стоил в 2012 г. всего 3 млн рублей. Исследования по СУБД *Clusterix-N, Clusterix-G* и *PerformSys* продолжаются.

Литература

1. Szalay A.S. The Sloan Digital Sky Survey and beyond //SIGMOD Record. 2008. Vol.37, No.2. P. 61–66.
2. Shiers J. The Worldwide LHC Computing Grid (worldwide LCG) //Computer Physics Communications. 2007. Vol. 177 No. 1-2. P. 219–223.
3. Taniar D., Leung C., Rahayu J. W. High-performance parallel database processing and grid databases. – John Wiley & Sons Inc., Hoboken, 2008.
4. Raikhlin, V.A. Simulation of Distributed Database Machines //Programming and Computer Software. Vol. 22, Issue 2, 1996, P. 68-74.
5. Абрамов Е.В. Параллельная СУБД Clusterix. Разработка прототипа и его натурное исследование // Вестник КГТУ им. А.Н.Туполева. Казань. 2006. № 2. С.50-55.
6. Martin, J. Computer database organization. Second Edition – Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, 1977. 713 p.
7. Райхлин В.А., Минязев Р.Ш. Мультикластеризация распределенных СУБД консервативного типа // Нелинейный мир. Т.9. 2011. № 8. С.473-481.
8. Классен Р.К. Повышение эффективности параллельной СУБД консервативного типа на кластерной платформе с многоядерными узлами // Вестник КГТУ им. А.Н.Туполева. Казань. 2015. № 1. С. 112-118.
9. CoGaDB – Column-oriented GPU-accelerated DBMS. URL:http://cogadb.cs.tudortmund.de/wordpress.
10. PGStrom 2016. URL: https://wiki.postgresql.org/index.php?title=PGStrom&oldid=25517.
11. Беседин Д. Первый взгляд на DDR3. Изучаем новое поколение памяти DDR SDRAM, теоретически и практически //ixbt.com. 2007. URL: http://www.ixbt.com/mainboard/ddr3-rmma.shtml
12. Петров С.В. Шины PCI, PCI Express. Архитектура, дизайн, принципы функционирования. //СПб.: БХВ-Петербург, 2006. 416 с.
13. Rauhe H. Finding the Right Processor for the Job Co-Processors in a DBMS. Ilmenau University of Technology, Ilmenau, 2014. 113 p.
14. Wenbin F., Bingsheng H., Qiong L. Database Compression on Graphics Processors //Proc. VLDB Endow., Vol. 3, No. 1-2, Sep 2010. P.670-680.
15. 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. P. 1-5. DOI: 10.1109/SIBCON.2017.7998474
16. Blelloch G. Introduction to Data Compression. Pittsburgh: Carnegie Mellon University, 2013. P.25-26
17. Классен Р.К. Программа региональной балансировки нагрузки к базе данных консервативного типа на кластерной платформе «PerformSys». Свидетельство о государственной регистрации программы для ЭВМ №2017611785 от 09.02.2017.
18. Монтируемый в стойку сервер System x3950 X6 | Lenovo (RU). URL: https://www3.lenovo.com/ru/ru/datacenter/servers/mission-critical/System-x3950-X6/p/WMD00000002

Райхлин Вадим Абрамович. Профессор кафедры Казанского национального исследовательского технического университета им. А.Н. Туполева. Окончил Казанский авиационный институт им. А.Н. Туполева в 1962 году. Доктор физико-математических наук, профессор. Количество печатных работ: 101, в том числе 12 монографий. Область научных интересов: конструктивное моделирование систем. E-mail: no-form@evm.kstu-kai.ru

Классен Роман Константинович. Аспирант. Казанского национального исследовательского технического университета им. А.Н. Туполева (КАИ). Окончил КАИ в 2015 году. Количество печатных работ: 8. Область научных интересов: высокопроизводительные системы, big data, информационные технологии. E-mail: klassen.rk@gmail.com

Relatively inexpensive hybrid technology of large volumes conservative DBMS

V. A. Raikhlin, R. K. Klassen

In article are discussed the issues of organization of conservative DBMS on comparatively inexpensive cluster platforms using MySQL and GPU-accelerators at the executive level. The relevance of the adopted orientation for work with large-scale databases is determined by current trends in the intellectual processing of large information arrays. Increasing the size of databases requires them to be hashed over cluster nodes. This necessitates the use of a regular query plan. The use of homogeneous cluster technologies (DBMS Clusterix with the optimal configuration "combined symmetry") requires an additional dynamic segmentation of intermediate and temporal relationships. Hybrid technologies (Clusterix-N and Clusterix-G projects) are proposed for managing large-scale databases, which allows excluding dynamic segmentation and significantly improving efficiency by the criterion of productivity / cost. In them, the data is located in nodes RAM. A distinctive feature of the Clusterix-G DBMS with GPU accelerators is the work with compressed databases, which allows increase their data volume on nodes with limited amount of memory. The proposed technologies are more efficient in comparison with Clusterix, and the Clusterix-G performance should exceed Clusterix-N for medium speed interconnect.

Keywords: Conservative DBMS. Big data. Affordable cluster platforms. Regular query plan. Hybrid technologies. Databases compression. Application of graphic accelerators. Comparative assessments of efficiency and productivity.

References

1. Szalay A.S. The Sloan Digital Sky Survey and beyond //SIGMOD Record. 2008. Vol.37, No.2. P. 61–66.
2. Shiers J. The Worldwide LHC Computing Grid (worldwide LCG) //Computer Physics Communications. 2007. Vol. 177 No. 1-2. P. 219–223.
3. Taniar D., Leung C., Rahayu J. W. High-performance parallel database processing and grid databases. – John Wiley & Sons Inc., Hoboken, 2008.
4. Raikhlin, V.A. Simulation of Distributed Database Machines //Programming and Computer Software. Vol. 22, Issue 2, 1996, P. 68-74.
5. Abramov E.V. Parallel'naya SUBD Clusterix. Razrabotka prototipa i ego naturnoe issledovanie [Parallel DBMS Clusterix. Development of prototype and its full-scale studits] // Herald of KSTU named after A.N. Tupolev. Kazan. 2006. № 2. C.50-55.
6. Martin, J. Computer database organization. Second Edition – Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632, 1977.
7. Raikhlin, V.A., Minjazev R.Sh. Mul'tiklasterizaciya raspredelennyx SUBD konservativnogo tipa [Multiclusterization of distributed DBMS of conservative type] // Nonlinear world. Vol.9. 2011. No.85. P.473-481.
8. Klassen R.K. Povyshenie e'ffektivnosti parallel'noj SUBD konservativnogo tipa na klasternoj platforme s mnogoyadernymi uzlami [Improving efficiency of parallel conservative type DBMS on a cluster platform with multi-core nodes] //Herald of KSTU named after A.N. Tupolev. 2015. No.1. P.112-118.
9. CoGaDB – Column-oriented GPU-accelerated DBMS. URL:http://cogadb.cs.tudortmund.de/wordpress.
10. PGStrom 2016. URL: https://wiki.postgresql.org/index.php?title=PGStrom&oldid=25517.
11. Besedin D. Pervyj vzglyad na DDR3. Izuchaem novoe pokolenie pamyati DDR SDRAM, teoreticheski i prakticheski [First look at DDR3. We study the new generation of DDR SDRAM memory, in theory and practically] //ixbt.com. 2007. URL: http://www.ixbt.com/mainboard/ddr3-rmma.shtml
12. Petrov S.V. Shiny PCI, PCI Express. Arxitektura, dizajn, principy funkcioniro-vaniya. [PCI bus, PCI Express. Architecture, design, principles of functioning.] // SPb.: BXV-Peterburg, 2006. 321-322 c.
13. 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.

14. Wenbin F., Bingsheng H., Qiong L. Database Compression on Graphics Processors //Proc. VLDB Endow., Vol. 3, No. 1-2, Sep 2010. P.670-680.
15. 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. P. 1-5. DOI: 10.1109/SIBCON.2017.7998474
16. Blelloch G. Introduction to Data Compression. Pittsburgh: Carnegie Mellon University, 2013. P.25-26.
17. Klassen R.K. Programma regional'noj balansirovki nagruzki k baze dannyx konser-vativnogo tipa na klasternoj platforme «PerformSys». [The program for regional load balancing to a conservative type database on the cluster platform «PerformSys».] Svidetel'stvo o gosudar-stvennoj registracii programmy dlya E'VM №2017611785 ot 09.02.2017. [Certificate of state registration of the computer program No. 2017611785 of 09.02.2017]
18. System x3950 X6 Rack Server | Lenovo (RU) URL: https://www3.lenovo.com/ru/ru/data-center/servers/mission-critical/System-x3950-X6/p/WMD00000002

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

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