МОГУТ ЛИ GPU-АКСЕЛЕРАТОРЫ СУЩЕСТВЕННО ПОВЫСИТЬ ЭФФЕКТИВНОСТЬ КОНСЕРВАТИВНЫХ СУБД ЗНАЧИТЕЛЬНЫХ ОБЪЕМОВ НА КЛАСТЕРНОЙ ПЛАТФОРМЕ?

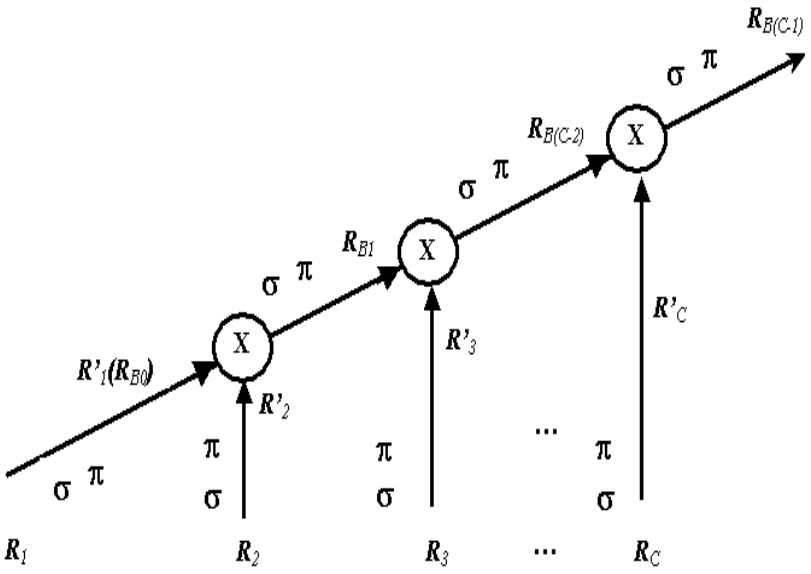
В.А. Райхлин, Р.К. Классен

ВВЕДЕНИЕ

В процессе доклада обсуждаются вопросы построения СУБД консервативного типа (с эпизодическим обновлением данных в специально выделяемое время) на платформе GPU-кластеров при объемах баз данных $V_{БД}$ от 100GB до единиц TB. Их актуальность определяется современными тенденциями интеллектуальной обработки больших информационных массивов с применением графических ускорителей – GPU. Повышение объема баз данных требует их хеширования по узлам кластера. Это обуславливает необходимость использования регулярного плана обработки запросов [1], транслируемых к схеме:

СЕЛЕКЦИЯ (σ) – ПРОЕКЦИЯ (π) – СОЕДИНЕНИЕ (σ_θ (R x S))

ВВЕДЕНИЕ



Регулярный план обработки запросов

По условию соединение всегда естественное.

Используется стратегия «множество узлов кластера – на один запрос».

При этом база данных оказывается распределенной по узлам, что допускает работу с базами данных повышенных объемов $V_{БД}$.

Получение любых промежуточных $R_{i'}$ и временных $R_{B(i-2)}$ отношений происходит параллельно

ВВЕДЕНИЕ

До сих пор при работах с базами данных мало используется вычислительный потенциал современных многоядерных процессоров. Примером может служить ранняя разработка СУБД Clusterix [2] и ее мультикластерная модификация Clusterix-M [3] на SUN-кластере с многоядерными узлами. Если объемы баз данных не слишком велики ($V_{БД}$ ограничен размерами оперативной памяти узла), то многоядерность узлов позволяет по иному подойти к организации параллельной СУБД, используя стратегию «один узел кластера на множество запросов» [4] . Что обеспечивает 100% загрузку всех ядер при специальной настройке инструментальной СУБД MySQL 5.6. В итоге получено серьезное повышение производительности кластера в сравнении с Clusterix-M.

ВВЕДЕНИЕ

Графические ускорители (GPU) уже применяются для ускорения работы СУБД CoGaDb [5], как расширение PG-Storm [6] для Postgres и др. Использование GPU позволяет существенно снизить время выполнения отдельных операций. При $V_{БД} \leq 3GB$ вся БД может храниться в глобальной памяти GPU (6GB для GPU Fermi). Тогда выполнение сервисных функций (прием запросов, их синтаксический анализ и др.) возлагается на CPU, а обработкой запросов будет заниматься только GPU. Эффективность такой СУБД может быть достаточно высока. При значительных $V_{БД}$ необходимость обмена данными с GPU существенно снижает производительность.

ВВЕДЕНИЕ

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 [7], в то время как для шины PCI-e 2.0x16 – 6,4 GB/s [8]. Именно по этой причине достигнутый в работе Rauhe H. «Поиск правильного процессора для работы как сопроцессор в СУБД» (Finding the Right Processor for the Job Co-Processors in a DBMS) [9] рост производительности сервера БД от использования GPU при обработке достаточно простых одиночных запросов не превысил 40%.

Постановка задачи

Работа по схеме «один узел кластера на множество запросов», сама по себе, уже обеспечивает достаточно высокую производительность при ограниченных $V_{БД}$. Поэтому одна из рассматриваемых нами задач: позволит ли использование GPU-ускорителей существенно превысить достигнутое в [4] ускорение для баз данных с $V_{БД} \leq 100GB$ (по условию объем оперативной памяти узла не превышает 128GB)?

Необходимое сжатие данных перед их отсылкой в GPU может быть достигнуто путем выполнения операций «select-project» на CPU (операции «join» реализуются на GPU) либо хранением БД на дисках в сжатом виде и реализацией операций «select» на GPU (операции «project – join» реализуются на CPU).

Постановка задачи

Для получения нужного эффекта в первом варианте необходимо удовлетворить следующие требования:

1. Реализация связок «*select – project*» на CPU должна выполняться значительно быстрее обработки запроса в целом инструментальной СУБД без акселерации и совмещаться во времени с передачей данных по шине PCI-e.
2. Коэффициент сжатия информации перед ее отсылкой в GPU должен быть значителен, так чтобы время передачи по шине PCI-e удовлетворяло требованию совмещения условия 1, а объем глобальной памяти GPU оказался достаточным для реализации операции *«join»*.

Постановка задачи

Во втором варианте для получения ответа на поставленный вопрос достаточно ограничиться требованием значительно более быстрой реализации на CPU операции «join» по сравнению с обработкой запроса в целом инструментальной СУБД без акселерации.

Еще одна задача связана с обработкой баз данных достаточно больших VБД – от сотен GB до единиц TB. Здесь уже, аналогично Clusterix, необходимо применять хеширование БД на множестве узлов кластера, но с ее предварительным сжатием. Суть вопроса в данном случае: может ли использование GPU существенно ускорить работу СУБД по сравнению с СУБД Clusterix-M? Рассматривается гибридный вариант: операции «select – project» выполняются на узлах IO с ускорителями, операции «join» – на узлах JOIN без GPU

Два вопроса ускорения работы СУБД

1. Позволит ли использование GPU-ускорителей существенно превысить достигнутое ускорение для баз данных с VБД ≤ 100GB (по условию объем оперативной памяти узла не превышает 128GB)?

(схема один узел кластера на множество запросов)

2. Может ли использование GPU существенно ускорить работу СУБД по сравнению с СУБД Clusterix-M?

(схема множество узлов кластера на один запрос)

Сравнение схем организации работы

Множество узлов кластера на один запрос

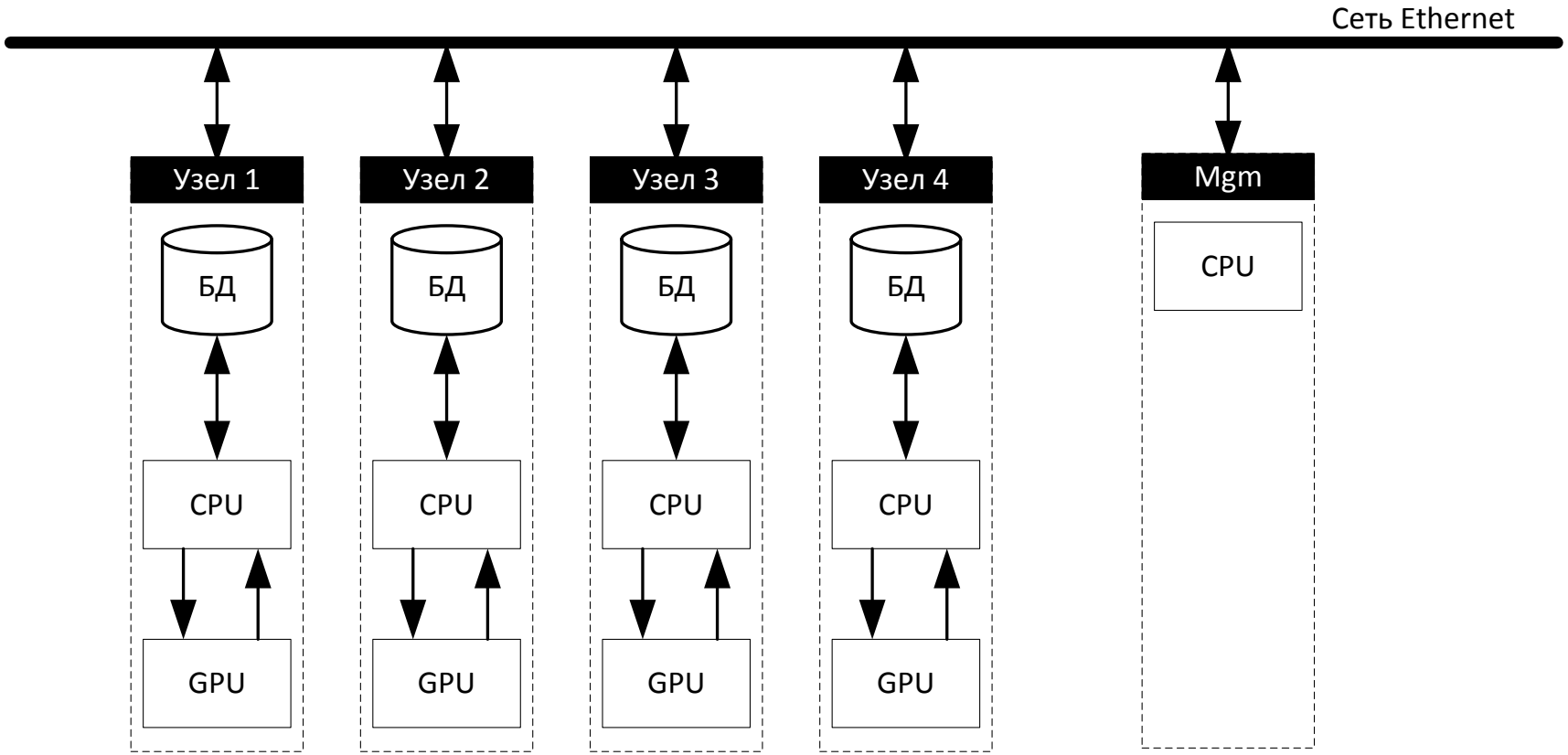
- БД распределяется по узлам монокластера
- Падение скорости обработки из-за барьерной синхронизации
- Эффективно работает с БД превышающими объем ОЗУ одного узла
- Не полностью использует ресурсы вычислительного узла

Один узел кластера на множество запросов

- БД содержится на каждом узле в полном объеме
- Барьерная синхронизация отсутствует
- Резкое падение производительности при работе с БД объемом превышающим объем ОЗУ одного узла
- Обеспечивает полную загрузку ядер вычислительного узла

Позволит ли использование GPU-ускорителей существенно превысить достигнутое ускорение для баз данных с $V_{БД} \leq 100GB$ (по условию объем оперативной памяти узла не превышает 128GB)?

Гипотетическая система. $V_{БД}$ – до 100GB



Гипотетическая система. $V_{БД}$ – до 100GB

Гипотетическая система повторяет стратегию «узел на множество запросов» (в узле: 2 процессора, N –общее количество ядер), но с некоторыми отличиями. Теперь один процессор (N/2 ядер) – Host, а другой (еще N/2 ядер) – собственно исполнительный. Host-процессор управляет выполнением операций на исполнительных ядрах, пересылкой данных по сети, работой графического ускорителя. Каждый узел имеет собственную очередь запросов длиной N/2 +1 и полную копию сжатой или несжатой БД. Помимо исполнительных узлов, в системе имеется управляющий узел Mgm. Он раздает задание на обработку, балансирует нагрузку между исполнительными узлами, получает результат исполнения запроса. Запросы обрабатываются на GPU последовательно, один за другим, а операции «select – project» (в варианте 1) либо «join» в варианте 2 – параллельно на CPU (1 ядро на запрос).

Сжатие данных путем выполнения операций «select-project» на CPU

При подсчете объема несжатых 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';
```

Подготовка эксперимента

Эксперимент проводился на базе вычислительного узла с следующими характеристиками: Quad-core Intel Core i5-4670K CPU/2,5GHz/24GB RAM (DDR3-1600 в двухканальном режиме), 64-битная ОС Windows 10, дисковая подсистема узла – SSD 120GB с пропускной способностью 450 MB/s, GPU – Nvidia GTX 770 (с объемом памяти 2GB GDDR5).

В качестве инструментальной СУБД использовалась MySQL 5.7

Путем претрансляции, согласно регулярному плану обработки, для 14 отобранных запросов теста TPC-H без операций записи были сформированы загрузочные модули «select – project» (по каждому Ri) и «join» (по каждому запросу в целом). При этом все операции «project» были «опущены вниз». Объем тестовой БД составил 1GB. Она предварительно загружалась (со сжатием или без него) в оперативную память.

Объемы данных для обработки запросов TPC-H

№ 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

Предварительное сжатие БД

По условию сжатая база данных сохраняется в оперативной памяти узла. Разжатие данных вносит дополнительную задержку в обработку данных на GPU. Эта задержка частично нивелирует выигрыш во времени передачи. Различают СУБД, ориентированные на хранение данных по строкам и по столбцам. Лучшие показатели по сжатию имеют СУБД второго типа [10]. 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.

Алгоритм подготовки данных для сжатия

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

Обработка запроса

Запрос разбивается на подзапросы согласно регулярному плану обработки. Для каждой таблицы, участвующей в обработке, проецируется набор колонок, необходимых для выполнения операций «select». В графические укорители GPU 1 и GPU 2 поблочно передаются колонки одной таблицы, над ними производятся необходимые операции. Результат возвращается в хост-память для последующего выполнения операции «join». 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.

Подготовка эксперимента 2

Эксперимент проводился на той же платформе, что и эксперимент 1.

Подготовлены блоки данных различного размера, сжатых по алгоритму RLE.

Цель эксперимента: сравнение времен, затрачиваемых на простое копирование, с суммой времен, необходимых для копирования данных, сжатых по алгоритму RLE (Run Length Encoding) [11], и их разжатия в GPU.

Размер (Мбайт)	Копиро-вание, мс	Копирование и восстановление сжатой информации, мс		Увеличение эффективности передачи, %	
		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%
49,97	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%

Сравнительные оценки времени выполнения операций «select-project» и «join».

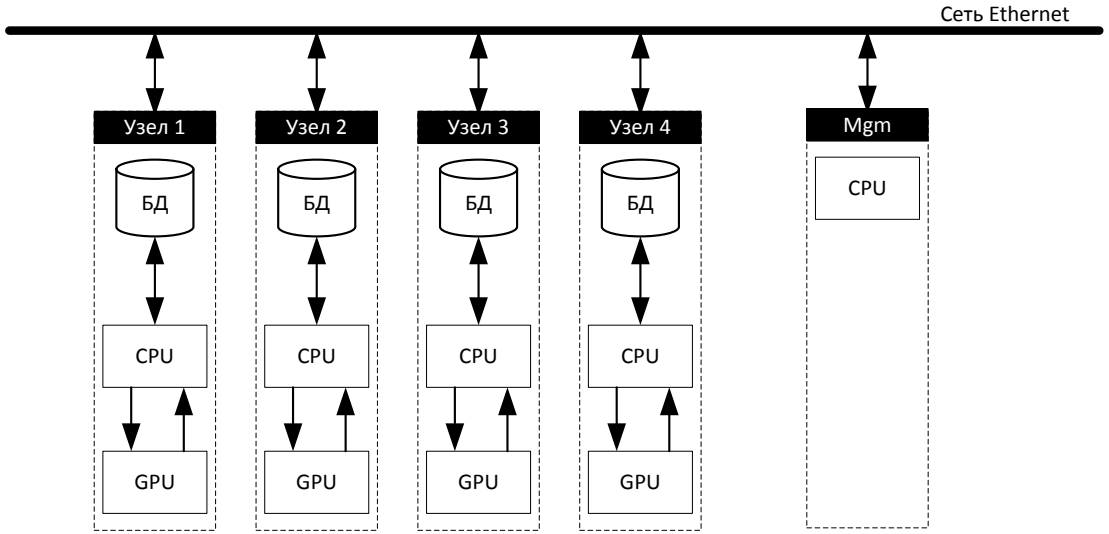
Сравнивались времена, затрачиваемые на реализацию всех операций «select-project» и единой операции «join» по каждому запросу ПТ, со временами с выполнением этих запросов в целом одним ядром под управлением СУБД MySQL 5.7 на той же платформе, что и в эксперименте 2. Эта процедура выполнялась по следующему алгоритму:

1. Создание промежуточных отношений в памяти MySQL для хранения результата «select-project».
2. Индексация промежуточных отношений.
3. Выполнение операции R1 join (join R2’(join R3’(…))) …).

№ запроса	Время обработки запроса, сек					Отношение времен обработки	
	Оригинального запроса	Выполнение «select-project»	Индексация рез-та «select-project»	Выполнение «join»	t_{Σ}^{join}		
	$t_{общ}$	$t_{\Sigma}^{\sigma,\pi}$	$t_{инд}$	t^{join}	$t_{инд} + t^{join}$	$t_{общ} / t_{\Sigma}^{\sigma,\pi}$	$t_{общ} / t_{\Sigma}^{join}$
1	21,579	9,808	0,000	0,000	0,000	2,20	нет «join»
2	0,145	2,050	3,633	0,011	3,644	0,07	0,040
3	2,142	7,939	3,270	0,163	3,433	0,27	0,624
4	1,035	6,668	1,623	0,470	2,093	0,16	0,495
5	1,123	8,721	11,892	0,154	12,046	0,13	0,093
6	2,846	5,054	0,000	0,000	0,000	0,56	нет «join»
7	0,901	7,624	5,506	0,261	5,767	0,12	0,156
8	3,335	9,837	22,394	0,081	22,475	0,34	0,148
9	5,315	11,355	26,805	2,234	29,039	1,00	0,183
10	1,850	6,707	1,614	0,944	2,558	0,28	0,723
11	0,229	1,896	1,209	0,069	1,278	0,12	0,179
12	3,760	6,561	0,809	0,075	0,884	0,57	4,253
13	3,302	1,833	0,714	4,703	5,417	1,80	0,610
14	3,270	5,339	0,198	0,264	0,462	0,61	7,078
Итого	50,830	91,392	79,667	9,428	89,096	0,556	0,571

Ответ на поставленный вопрос

Таким образом, необходимые условия эффективности для обоих вариантов использования GPU по схеме рисунке (условие 1 в варианте 1) не выполняются. Поэтому наш ответ на поставленный вопрос в рассмотренном случае отрицательный.

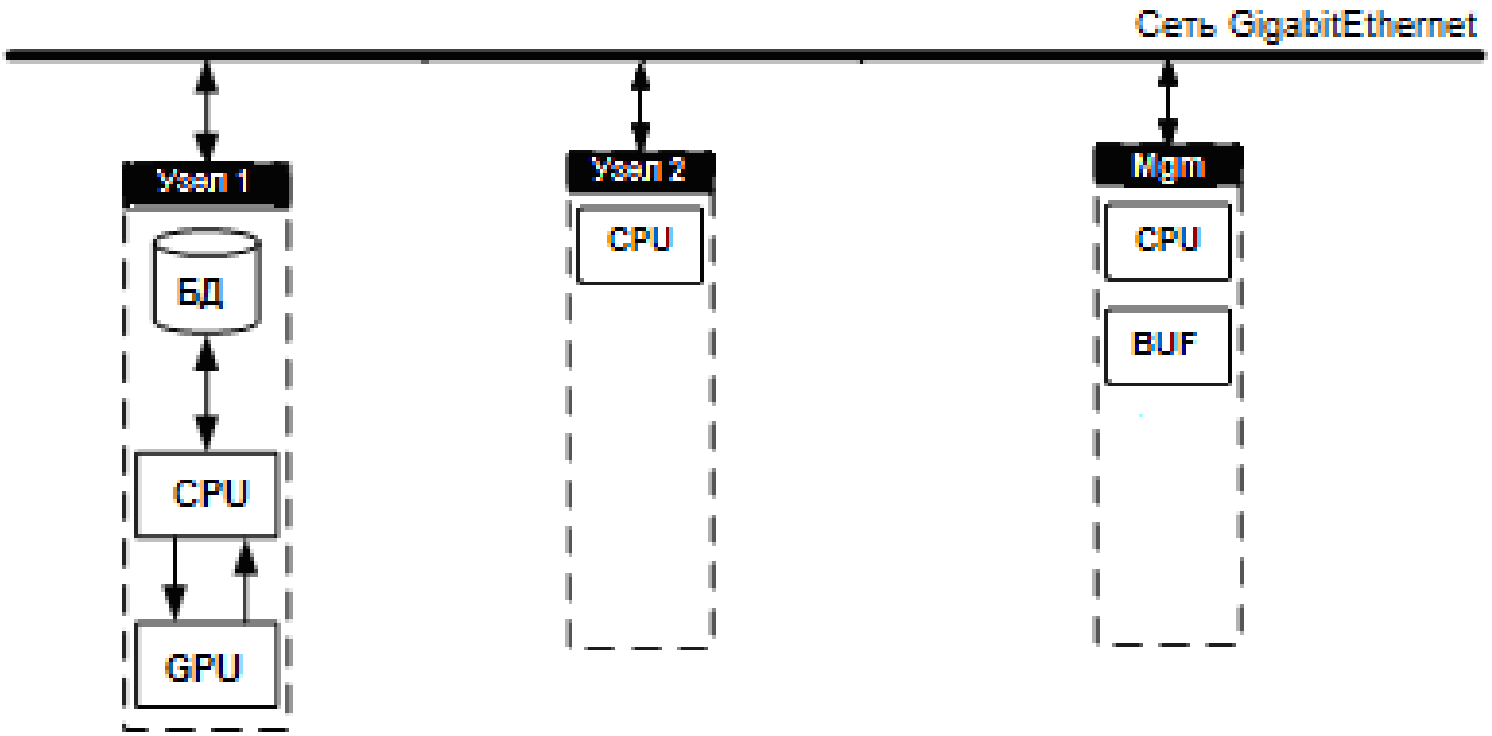


Может ли использование GPU существенно ускорить работу СУБД по сравнению с СУБД Clusterix-M?

Анализ для простейшего случая

Рассмотрим пример обработки системой с двумя исполнительными узлами (узлы 1 – IO и 2 – JOIN) и одним управляющим (Mgm) представительского теста ПТ из 14 запросов в случае $V_{БД}$= 1GB. Все узлы – 12-ядерные. Узел IO (с GPU) выполняет *«select–project»*-обработку запросов над сжатой БД. Узел JOIN – единые процедуры «*join»* по каждому запросу ПТ, параллельно по 12 запросам (ядро на запрос).

Анализ для простейшего случая



Анализ для простейшего случая

При этом суммарный объем отношений, задействованных в процессе выполнения ПТ, $\boldsymbol{V_{общ}^{\Sigma} \approx 9{,}7GB}$. В результате сжатия исходной информации после *«select–project»*-обработки остается $\boldsymbol{(\sigma, \pi)_{\Sigma} \approx 1{,}4GB}$ с коэффициентом сжатия $\boldsymbol{V_{общ}^{\Sigma}/(\sigma, \pi)_{\Sigma} \approx 6{,}87}$. Тогда для несжатой БД общий объем информации, передаваемой при обработке ПТ в IO-узле от CPU к GPU и обратно, составит $\boldsymbol{[V_{общ}^{\Sigma} + (\sigma, \pi)_{\Sigma}] \approx 11{,}1GB}$. Поскольку предполагается сравнение результатов теоретического анализа с полученными ранее для СУБД *Clusterix-M,* то рассматривается использование сети GigabitEthernet. Возможными перегрузками по интерконнекту (когда время ожидания передачи по сети для некоторых сообщений превышает значение тайм-аута, что приводит к «зависанию» системы) будем пренебрегать. Полагаем дополнительно, что этапы обработки ПТ в целом (*«select–project»*-обработка → передача ее результата с уровня IO к Mgm и далее – от Mgm на уровень JOIN → собственно *«join»*-обработка) следуют друг за другом.

Анализ для простейшего случая

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 в каждом IO-узле от CPU к GPU объемом $\boldsymbol{V_{общ}^{\Sigma}}$ составит $\boldsymbol{t_{п1}}$ **= 9,7GB/(3,2GB/s) = 3,03сек**, а от GPU к CPU – $\boldsymbol{t_{п2}}$ **= 1,4GB/(3,2GB/s) = 0,44сек**. Как нами установлено ранее, хранение исходной БД в сжатом виде с коэффициентом сжатия K=5 уменьшает $t_{п1}$ примерно на 40%. Поэтому для 14 запросов ПТ получаем $\boldsymbol{t_{п}'}$ **= 3,03·0,6 + 0,44 = 2,26сек**. Дополнительно необходимо учесть время на передачу данных по сети GigabitEthernet от IO к Mgm $\boldsymbol{t_{п}''}$ **=** $\boldsymbol{(\sigma, \pi)_{\Sigma} / \upsilon_{сети}}$ **=1,4GB/(0,1GB/s) = 14сек**. В итоге получаем $\boldsymbol{(t_{\Sigma}^{\sigma,\pi})'}$ **=** $\boldsymbol{t_{п}'}$ **+** $\boldsymbol{t_{п}''}$ **= 16,26сек**.

Время передачи информации

$V_{общ}^{\Sigma} \approx 9{,}7GB$

$(\sigma, \pi)_{\Sigma} \approx 1{,}4GB$

$V_{общ}^{\Sigma}/(\sigma, \pi)_{\Sigma} \approx 6{,}87$

$[V_{общ}^{\Sigma} + (\sigma, \pi)_{\Sigma}] \approx 11{,}1GB$

$t_{п1} = 9{,}7GB/(3{,}2GB/s) = 3{,}03сек$

$t_{п2} = 1{,}4GB/(3{,}2GB/s) = 0{,}44сек$

$t_{п}' = 3{,}03 \cdot 0{,}6 + 0{,}44 = 2{,}26сек$

$t_{п}'' = (\sigma, \pi)_{\Sigma} / \upsilon_{сети} = 1{,}4GB/(0{,}1GB/s) = 14сек$

$(t_{\Sigma}^{\sigma,\pi})' = t_{п}' + t_{п}'' = 16{,}26сек$

Анализ для простейшего случая

Полагаем, что выполнение единых по запросам операций *«join»* начинается по окончании всех сетевых передач для пакета из 14 запросов непрерывного потока. При оценке общего времени на *«join»*-обработку $(t_{\Sigma}^{join})'$ будем учитывать время передачи $t_{п}'''$ накопленного в BUF пакета R_i' от Mgm на уровень JOIN 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. Суммарный объем R_i' по этим запросам равен 135,6 МВ. Поэтому $t_{п}''' = t_{п}'' -$ 1,4сек = 12,6сек

$$t_{п}''' = t_{п}'' - 1{,}4\text{сек} = 12{,}6\text{сек}$$

Анализ для простейшего случая

Пакет из оставшихся 12 запросов будет параллельно обрабатываться на 12 ядрах узла JOIN. Время *«join»*-обработки этого пакета $(t_{\Sigma}^{join})^{12}$ определится самой медленной процедурой. Последней завершится обработка по запросу №9, т.е. $(t_{\Sigma}^{join})^{12} \approx 29{,}04$сек. Общее время $(t_{\Sigma}^{join})' = t_{п}'''' + (t_{\Sigma}^{join})^{12} = 12{,}6$сек + 29,04сек = 41,64сек. Время обработки системой пакета из 14 запросов $T = (t_{\Sigma}^{\sigma,\pi})' + (t_{\Sigma}^{join})' = 16{,}26$сек + 41,64сек =57,9 сек.

$(t_{\Sigma}^{join})^{12} \approx 29{,}04$сек

$(t_{\Sigma}^{join})' = t_{п}'''' + (t_{\Sigma}^{join})^{12} = 12{,}6$сек + 29,04сек = 41,64сек

$T = (t_{\Sigma}^{\sigma,\pi})' + (t_{\Sigma}^{join})' = 16{,}26$сек + 41,64сек =57,9 сек

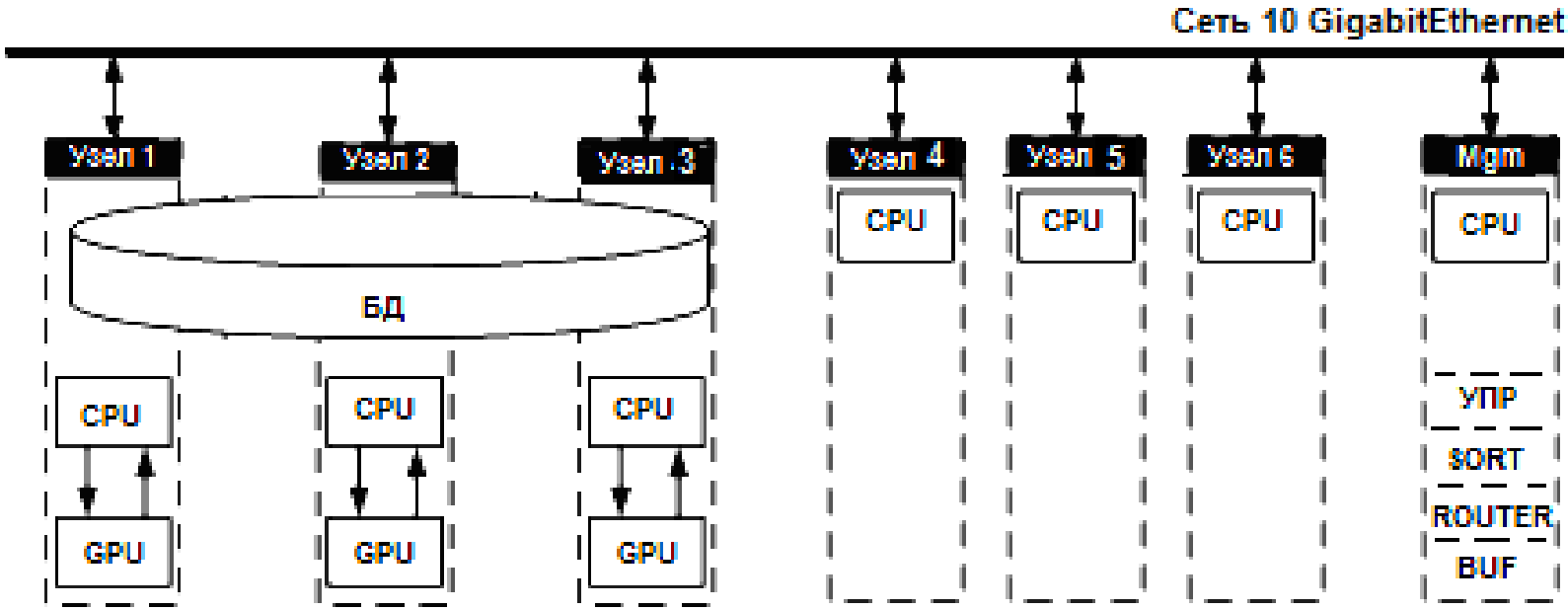
Ответ на поставленный вопрос

СУБД *Clusterix-M* на платформе SUN-кластера с 6-ю задействованными исполнительными узлами (и одним управляющим) в сети GigabitEthernet показала [3] при $V_{БД}$ = 1GB время обработки ПТ ≈ 201сек >> Т. Даже принимая во внимание, что исходная БД при проведении эксперимента с *Clusterix-M* находилась на диске, приходим к выводу: *при обработке пакета 14 запросов ПТ и сравнительно небольших $V_{БД}$ рассмотренная простейшая система должна быть значительно эффективнее Clusterix-M*.

Принятая к разработке натурная модель Clusterix-G

организуется на платформе GPU-кластера КНИТУ-КАИ с шестью исполнительными узлами (узлы 1-6) и одним управляющим (Mgm). 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-массив 4 WD1000 DHTZ/1TB. Операционная система – SUSE Linux Enterprise Server версии 11. Интерконнект между узлами – GigabitEthernet с 24-портовым коммутатором SSE G24-TG4 / 10 GigabitEthernet с 8-портовым коммутатором NETGETAR XS708T-100NES. Узлы 1-3 – IO. Узлы 4-5 – JOIN. Акселераторы в них не используются. На все узлы, кроме IO, установлена СУБД MySQL 5.7.

Принятая к разработке натурная модель Clusterix-G



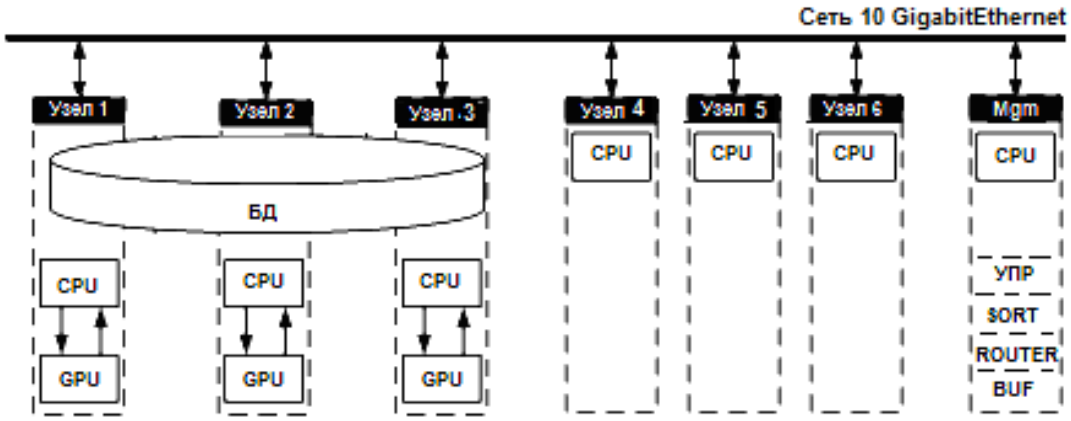
Принятая к разработке натурная модель Clusterix-G

Сравнительно небольшие размеры используемого GPU-кластера ограничивают возможности проведения исследований на его основе случаем $V_{БД} \leq 200GB$. Причина в следующем. Согласно таблице результатов 1 эксперимента, суммарный объем промежуточных отношений ПТ $\approx 1{,}4\ V_{БД}$. При $V_{БД} = 200GB$ это дает 280GB. Данные такого объема (но не более!) еще можно разместить в оперативной памяти трех узлов JOIN (по условию один узел «принимает» не более 100GB). Из той же таблицы следует, что суммарные объемы R_i' для разных запросов ПТ с «*join*»-обработкой сильно различаются ~ от $0{,}0065V_{БД}$ до $0{,}428V_{БД}$. Так что при действии непрерывного потока запросов, случайно формируемых на множестве запросов ПТ, в каждый момент времени в одном JOIN-узле могут обрабатываться от одного до десяти запросов.

Принятая к разработке натурная модель Clusterix-G

По аналогии с ранее разработанными СУБД *Clusterix* и мультикластерной СУБД *Clusterix-M*, для целей разработки натурной модели *Clusterix-G* принят следующий ориентировочный план ее функционирования.

Модуль 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.

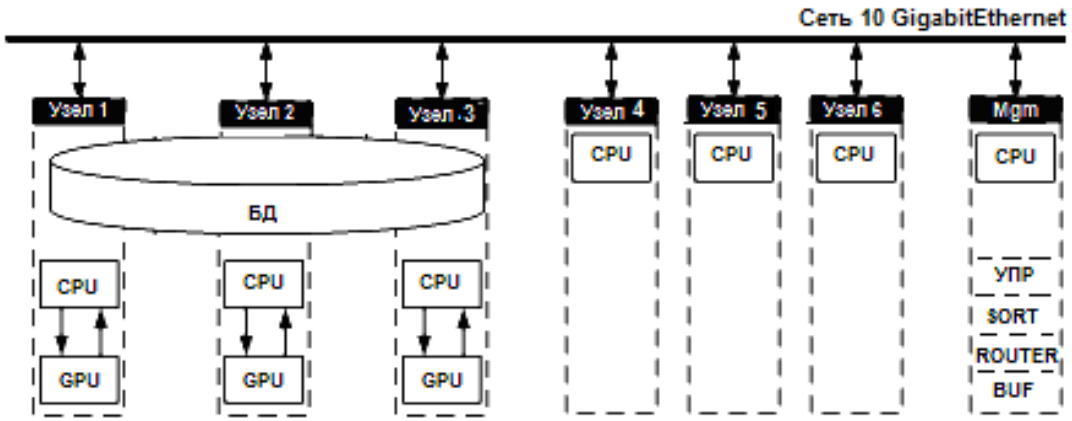


Принятая к разработке натурная модель Clusterix-G

Модуль SORT выполняет функции агрегации и сортировки результата обработки запроса.

Модуль ROUTER – функцию балансировки нагрузки между узлами JOIN.

В буфере BUF по каждому запросу осуществляется конкатенация частей отношений R_i' , подлежащих дальнейшему соединению. Функции управления реализуются по сети GigabitEthernet



Принятая к разработке натурная модель Clusterix-G

На уровне IO последовательно выполняются операции *«select – project»* по каждому запросу. По условию все операции «*project»* «опущены вниз». CPU в IO проецируют колонки, формируют блоки сжатой БД, управляют очередью своих команд, работой графических ускорителей (2 GPU на узел), пересылкой данных по шине PCI-e и сети 10 GigabitEthernet. 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.

Принятая к разработке натурная модель Clusterix-G

Дадим ориентировочные оценки для случая обработки натурной моделью пакета запросов ПТ при $V_{БД}$ = 200GB. Согласно таблице результатов эксперимента 1 будем полагать, что один из узлов JOIN выделяется исключительно для обработки запроса №4, а два других – для обработки остальных 11 запросов с *«join»*. Учтем дополнительно следующее. Дуплексный характер сети передачи данных позволит организовать совмещение передач от IO к Mgm и от Mgm к JOIN. Аналогичный дуплекс по шине PCI-e и использование двух ускорителей в каждом IO-узле позволит совместить во времени разжатие, обработку порции данных и передачу результата host-процессору в одном GPU с передачей очередной порции другому GPU. Тогда время собственно *«select–project»*-обработки определится односторонним обменом CPU →GPU по шине PCI-e.

Принятая к разработке натурная модель Clusterix-G

Соответственно, при условии линейных зависимостей от $V_{БД}$ и в прежних обозначениях: $t_п' \approx t_{п1} = 121{,}2$сек; $t_п'' \approx 280$сек и $(t_\Sigma^{\sigma,\pi})' = t_п' + t_п'' \approx 0{,}11$ час. Учитывая, что $(t_\Sigma^{join})^{12}$ определяется временем *«join»*-обработки запроса №4, имеем: $(t_\Sigma^{join})' = (t_\Sigma^{join})^{12} \approx 1{,}61$ час. Поскольку $(t_\Sigma^{join})' >> (t_\Sigma^{\sigma,\pi})'$, то общее время T обработки ПТ в целом $T = (t_\Sigma^{\sigma,\pi})' + (t_\Sigma^{join})' \approx (t_\Sigma^{join})'$. Поэтому в блоке IO целесообразно оставить лишь один узел, дополнив блок JOIN до пяти узлов. Тогда $(t_\Sigma^{\sigma,\pi})' \approx 0{,}18$час, что незначительно повлияет на величину T.

$t_п' \approx t_{п1} = 3{,}03\text{сек}\cdot 0{,}6\cdot 200/3 = 121{,}2\text{сек}$
$t_п'' \approx 1{,}4\text{GB}\cdot 200/(1\text{GB/s}) = 280\text{сек}$
$(t_\Sigma^{\sigma,\pi})' = t_п' + t_п'' \approx 401{,}2\text{сек} \approx 0{,}11$ час
$(t_\Sigma^{join})' = (t_\Sigma^{join})^{12} = 200\cdot 29{,}04 = 5808\text{сек} \approx 1{,}61$ час
$T = (t_\Sigma^{\sigma,\pi})' + (t_\Sigma^{join})' \approx (t_\Sigma^{join})'$

Принятая к разработке натурная модель Clusterix-G

Реальная динамика процессов в системе при действии непрерывного потока запросов, случайно формируемых на множестве запросов ПТ, будет существенно более сложной по сравнению с рассмотренной. При этом рост числа узлов JOIN поможет разгрузить BUF Mgm от накопления R_i' по ожидающим исполнения запросам и будет способствовать повышению производительности системы.

ОБСУЖДЕНИЕ

Перспективная стратегия «множество запросов на один узел кластера» с репликацией БД по отдельным узлам ориентирована на работу с консервативными базами данных умеренных объемов, не превышающих объем оперативной памяти узла. Переход к GPU-кластерам с использованием регулярного плана обработки запросов и адаптацией инструментальных СУБД к такой платформе в этом случае неконкурентоспособен. Целесообразна разработка специализированных СУБД с оптимизацией запросов, ориентированной на использование графических ускорителей. [12]

ОБСУЖДЕНИЕ

При использовании обычных вычислительных кластеров, организация обработки запросов к консервативным БД объемами до нескольких ТВ 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 указанных обстоятельств оказывается достаточно низкая производительность на грани масштабируемости.

ОБСУЖДЕНИЕ

Имеются веские основания полагать, что серьезное повышение эффективности в данном случае должно иметь место при переходе на платформу GPU-кластера при условии хранения распределенной БД в сжатом виде в оперативной памяти IO-узлов и организации обработки в JOIN-узлах по схеме «запрос на ядро». В IO-узлах GPU-акселерация используется для реализации операции *«select».* Экспериментальное подтверждение справедливости приведенного в статье обоснования эффективности такого перехода связывается с разработкой и исследованием натурной модели СУБД *Clusterix-G*.

ОБСУЖДЕНИЕ

Реализация по рассмотренному принципу GPU-СУБД объемами до нескольких ТВ требует построения кластеров с достаточно большим числом JOIN-узлов и скоростной сетью передачи данных. При этом необходимо выполнение условия: суммарный объем промежуточных отношений для любого запроса не должен превышать объема оперативной памяти узла. Использование перспективных технологий с объемами оперативной памяти до 2ТВ и значительным числом ядер (не менее 18) в узле делает подход «Один узел кластера на множество запросов» недосягаемым по эффективности другими известными методами построения большеобъемных консервативных СУБД кластерного типа. Во всяком случае, экспериментальных данных, опровергающих это утверждение, пока не имеется.

ОБСУЖДЕНИЕ

Представляет интерес вопрос: насколько все же оправдано использование GPU-узла для реализации *«select – project»*-обработки, и нельзя ли в данном случае применить схему «запрос на ядро» с хранением несжатой БД в оперативной памяти (ОП) узла? Поскольку по условию $V_{ОП}$ не позволяет загрузку данных объемом >100GB, придется БД хешировать по двум узлам, так что в блоке JOJN останется 4 узла. Но это не повлияет на значение $\mathbf{(t_{\Sigma}^{join})' = 1{,}61час}$. Время параллельного функционирования блока IO определится величиной $t_{\Sigma}^{\sigma,\pi}$ для самого «тяжелого» на ПТ запроса. Согласно таблице результатов 3 эксперимента, таковым вновь является запрос №9. Для него $\mathbf{t_{\Sigma}^{\sigma,\pi} = 11{,}355сек \cdot 100 = 1135{,}5сек \approx 0{,}32час}$, что сравнимо с полученным для одного GPU-узла значением **0,18час**.

ОБСУЖДЕНИЕ

Таким образом, из проведенного анализа следует: максимум, что может дать применение GPU-акселерация для натурной модели при действии непрерывного потока запросов, случайно формируемых на множестве запросов ПТ, это повышение производительности примерно на 25% за счет ввода в действие пятого узла JOIN. Безусловно, сделанный вывод требует экспериментальной проверки.

Выполненный анализ преследовал цель решения задачи более скромными средствами, доступными в настоящее время широкому кругу научно-образовательных организаций. В частности, GPU-кластер, на основе которого строится натурная модель, может функционировать и в мультизадачном режиме, когда каждому из шести пользователей отводится свой GPU-узел.

Список литературы

1. *Raikhlin, V.A.* Simulation of Distributed Database Machines //Programming and Computer Software. Vol. 22, Issue 2, 1996, P. 68-74.
2. *Abramov E.V.* Parallel DBMS Clusterix. Development of prototype and its full-scale studits //Herald of KSTU named after A.N. Tupolev. 2006. No.2. P.50-55.
3. *Raikhlin, V.A., Minjazev R.Sh.* Multiclusterization of distributed DBMS of conservative type //Nonlinear world. Vol.9. 2011. No.85. P.473-481.
4. *Klassen R.K.* 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.
5. CoGaDB – Column-oriented GPU-accelerated DBMS. URL:http://cogadb.cs.tudortmund.de/wordpress.

Список литературы

6. PGStrom 2016. URL: https://wiki.postgresql.org/index.php?title=PGStrom&oldid=25517.
7. 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 (дата обращения: 01.10.2016).ixbt.com. 2007.
8. Петров С.В. Шины PCI, PCI Express. Архитектура, дизайн, принципы функционирования. СПб.: БХВ-Петербург, 2006. 321-322 с.
9. *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.

Список литературы

10. *Wenbin F., Bingsheng H., Qiong L.* Database Compression on Graphics Processors //Proc. VLDB Endow., Vol. 3, No. 1-2, Sep 2010. P.670-680.
11. *Blelloch G.* Introduction to Data Compression. Pittsburgh: Carnegie Mellon University, 2013. P.25-26.
12. *Bres S.* Efficient query processing in co-processor-accelerated database //University of Magdeburg, 2015.