Clusterix-подобные СУБД консервативного типа класса BIG DATA

В. А. Райхлин, Р. К. Классен

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

Аннотация. Целесообразность разработок СУБД консервативного типа с эпизодическим обновлением данных определяется особенностями OLAP-технологий. Вопросы создания таких СУБД требуют серьезного обсуждения. В этом обзоре систематизированы основные результаты исследований научной группы Clusterix КНИТУ-КАИ по консервативным СУБД на базе вычислительных кластеров. Цель проведенных исследований актуальна: разработка подходов к синтезу сравнительно эффективных по критерию «производительность/стоимость» отечественных СУБД класса Big Data. Сравнение проводилось с лучшими зарубежными открытыми системами. Разрабатываемые СУБД доступны к применению организациям с ограниченными финансовыми возможностями. Должное внимание уделено элементам теории кластерных СУБД консервативного типа. Рассмотрены: базовые конфигурации систем Clusterix, динамика таких СУБД, эффекты их самоорганизации. За основу исследований взята методология конструктивного моделирования систем.

Ключевые слова: кластерные СУБД консервативного типа, элементы теории, базовые конфигурации, динамика процессов, эффекты самоорганизации, отечественные СУБД класса Big Data, сравнительная эффективность.

DOI 10.14357/20718632240304 **EDN** DDMDGU

Введение

Актуальность разработок СУБД консервативного типа (с эпизодическим обновлением данных) определяется тенденциями развития технологии аналитической обработки данных [1, 2].

Но вопросы динамики протекающих в них процессов и их целесообразной организации все еще недостаточно изучены. Данный обзор частично восполняет этот пробел. Он систематизирует многолетние исследования научной группы Clusterix КНИТУ-КАИ в области кластерных СУБД консервативного типа с регулярным планом [3] обработки запросов. Рассмотрение проводится с позиций конструктивного моделирования систем (КМС) [4].

Отметим главное в КМС. В условиях неполноты информации процесс синтеза рассматривается с системных позиций в предположении, что синтезируемый объект моделирует поведение некоторой гипотетической системы, заданной своим оператором назначения. Моделирование системы проводится в рамках соответствующей модели синтеза, или ***S***-модели (***S*** – от ***Synthesis***). Она строится эвристически, т.е. неформально. Процесс конструктивного моделирования итеративен и предполагает симбиоз трех равноправных компонент (Рис. 1).

Характерной особенностью ***S***-модели является постулирование выявленных в процессе КМС свойств множества эффективных реализаций системы как основ теории и предпосылки разработки конструктивного метода. Постулаты

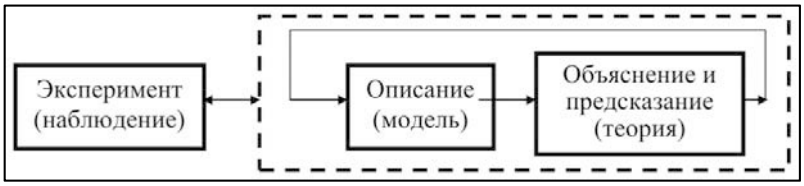


Рис. 1. Процесс конструктивного моделирования систем

рассматриваются как декларируемые закономерности, справедливые в меру накопленных знаний и опыта. В силу последнего система постулатов должна быть открытой и корректироваться с появлений новых знаний.

Объемы баз данных в десятки, сотни GB и более нередки для относительно небольших предприятий с ограниченными финансовыми возможностями. Коммерческие СУБД обладают высокой производительностью и надежностью, но чрезмерно большой стоимостью.

Удачной альтернативой дорогостоящим параллельным СУБД являются свободно распространяемые разработки с открытым исходным кодом Hadoop [5] и Spark [6]. Они высокопроизводительны, хорошо масштабируются, их требования к аппаратной платформе весьма скромные. Но это – зарубежные системы.

Чтобы не оказаться в догоняющей позиции, новые отечественные СУБД следует создавать на базе инструментальных СУБД с открытым кодом, поддерживаемых международным сообществом [7]. Этому требованию удовлетворяет отечественная разработка Postgres Pro [8]. Но она – одноузловая, а потому – недостаточно производительная. Целесообразна разработка элементов теории кластерных СУБД консервативного типа, что было сделано выявлением свойств базовых версий СУБД Clusterix небольшой производительности при ограниченных объемах данных. Исследования по СУБД класса Big Data развивались с ориентиром на размещение БД в суммарной оперативной памяти исполнительных узлов кластера, полную загрузку процессорных ядер и GPU-акселерацию.

1. Элементы теории кластерных СУБД консервативного типа

Исходные посылки. Для консервативных СУБД свойственна OLAP нагрузка [9], характеризующаяся высоким удельным весом сложных запросов типа «селекция (σ)– проекция (π) – соединение ($\otimes\,\sigma$)», $\otimes\,\sigma$ – декартово произведение, оперирующих множеством таблиц с большим числом операций соединения. По условию соединение всегда естественное [10].

Постулат 1 [11]: *Для параллельных СУБД консервативного типа регулярный план обработки запросов (Рис. 2), является предпочтительным*.

Согласно постулату, кластер консервативных баз данных включает: Host ЭВМ, процессоры нижнего IO_r и верхнего $Join_j$ уровней. Общее число физических процессоров в базовых системах Clusterix $N=h+1,\ h=2n,\ n=1,\ 2,\ldots$ Обозначим: m – число страниц базы данных, p и $q=m=h–p$ – числа процессоров $Join_j$ и I/O_r. Рассмотрение ограничено вариантами: $k^{-1} \in \{0,⅓,½,1\}$. Вариант $k^{-1}=0\ (p=0)$, когда функции IO_r и $Join_j\ (r=j)$ реализуются на одном процессоре, назван *линейкой*. Вариант $k=1$ – *симметрией*.

Есть еще три процессора. Процессор УПР реализует функции управления всеми процессорами системы. Процессор ПТР претранслирует исходный запрос к виду регулярного дерева. Процессор SORT, объединяет результаты на множестве узлов, выполняет операции агрегации (SUM, AVG, MAX, MIN и др.) и сортировки

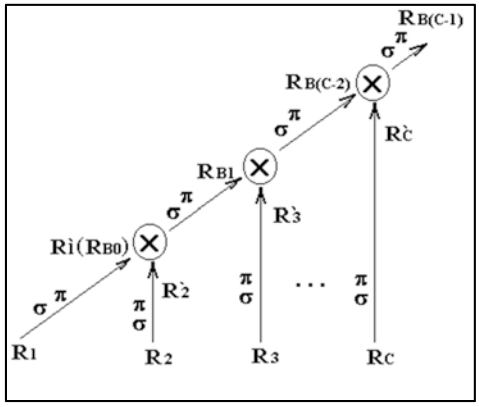


Рис. 2. Регулярный план обработки запросов

указанных результатов. Все три процессора функционируют на Host ЭВМ. Работа на уровне файловой системы, системных буферов, алгоритмов доступа к данным, работы с индексами и т.п. реализуется с помощью инструментальной СУБД MySQL. Для обеспечения устойчивого функционирования программной системы используется барьерная синхронизация [12].

В прототипе СУБД Clusterix [13] реализованы два вида параллелизма:

1) *Горизонтальный* – достигается параллельной обработкой несколькими процессорами одного уровня разных частей данных. Хеш-функция, использованная в Clusterix:

$$hash=((key\_field1 \bmod q) + (key\_field2 \bmod q) + \ldots + (key\_fieldP \bmod q)) \bmod q, \quad (1)$$

где P – количество полей в первичном ключе; $mod\ q$ – операция деления по модулю q. Как показали эксперименты, данный подход обеспечивает достаточно равномерное распределение данных по страницам.

2) *Вертикальный* – конвейерный механизм обработки запросов. Обусловлен обработкой запроса согласно регулярному плану.

Исходный SQL-запрос пользователя транслируется в пакет MySQL-фрагментов для процессоров IO и JOIN. В каждом такте конвейера совмещается выполнение по одному фрагменту того и другого. Совмещение как таковое имеет место только для архитектуры «симметрия». В случае «линейка» процессы IO и JOIN идут с разделением времени. В Табл. 1 дано краткое описание действий, выполняемых системой на каждом этапе.

Масштабируемость и производительность. Модельный эксперимент проводился с использованием натурной модели базовой

Табл. 1. Описание действий на каждом этапе

IO_EXEC	Исполнение операций π и σ над исходным отношением БД, получение промежуточного отношения R′ (сохраняется в главной памяти процессора IO)**.**
IO_HASH	Проведение операции хеширования (деление по модулю на число процессоров IO) над отношением R'_i по полю, участвующему в соответствующей операции соединения.
IO_WAIT_ LOAD	Пересылка частей отхешированных отношений R'_i между процессорами IO в соответствии с полученными на предыдущем этапе значениями хеш-функций и формирование итоговых отношении R'_i на каждом IO.
IO_WAIT_ SYNC	Ожидание, пока все модули IO выполнят текущую операцию (барьерная синхронизация модулей IO).
IO_NET_IO_ JOIN	Пересылка итоговых промежуточных отношений R'_i с процессоров IO на соответствующие процессоры JOIN.
JOIN_creating INDEX before JOIN	Создание индексов для промежуточного отношения R'_i, полученного на предыдущем такте работы IO.
JOIN_EXEC	Выполнение операции соединения над R'_i и временным отношением $R_{в(i-1)}$, полученным на предыдущем такте работы JOIN, формирование временного отношения $R_{в(i)}$.
JOIN_HASH	Проведение операции хеширования (деление по модулю на число процессоров JOIN) над отношением $R_{в(i)}$ по полю, участвующему в следующей операции соединения.
JOIN_WAIT_ LOAD	Пересылка отхешированных отношений $R_{в(i)}$ между процессорами JOIN (пересылка кортежей по сети и формирование итоговых отношении $R_{в(i)}$)
JOIN_WAIT_ SYNC	Ожидание, пока все модули JOIN выполнят текущую операцию (барьерная синхронизация модулей JOIN)
JOIN_creating INDEX after JOIN	Создание индексов для сформированных отношении $R_{в(i)}$.

СУБД Clustrerix [13] на трех различных платформах. Основной среди них является вычислительный кластер фирмы SUN из 22 узлов 2 Quad-core Intel Xeon E5450 CPU/1,87GHz/32GB. Интерконнект – GigabitEthernet/Infiniband 4X (20Gbps DDR) с коммутаторами Cisco. SAS диски XRB-SS2CD-146G10KZ с пропускной способностью 300 MB/s.

Мы стремились выявить некоторые общие закономерности поведения кластерных СУБД при изменении числа узлов и объемов баз данных для случаев одно- и двухпроцессорных (SMP) узлов. В качестве представительского теста (ПТ) взят ограниченный тест TPC-H из 14 запросов, не содержащих операций записи. За критерий эффективности принято отношение «производительность/стоимость».

По результатам работ [13-16], сформулирован постулат 2.

Постулат 2. *Для кластеров консервативных баз данных всегда существует граничное число страниц (число процессоров IO) $m=m_G$, которому отвечает максимум производительности на действующем ПТ. Это число зависит от используемой платформы, схемы БД, потока запросов и, при неизменных ПТ и схемы БД растет с увеличением объема $V_{БД}$. При работе на грани масштабируемости эффективность максимальна на «линейке».*

Вопросы динамической перестройки архитектуры кластера рассмотрены в работах [15, 16]. На Рис. 3 приведен график изменения времени выполнения ПТ (сек.) с ростом числа рабочих 2-процессорных SMP-узлов «линейки» в случае $V_{БД}$ =5,4GB. Видимый порог масштабируемости $m_G = h_G = 7\text{-}8$.

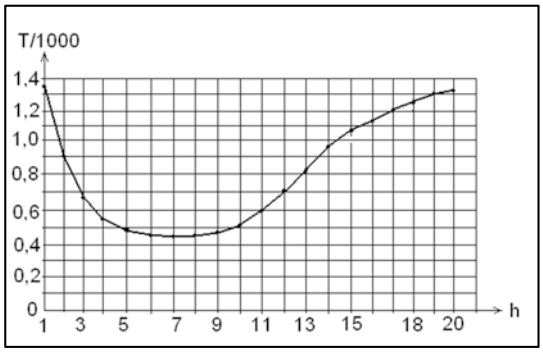
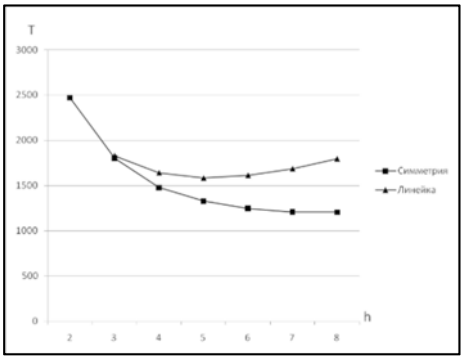


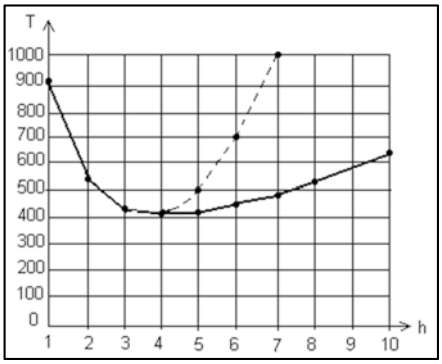
Рис. 3. SUN-кластер, $V_{БД}$ =5,4GB. Конфигурация «линейка»

Наличие двух процессоров в узле позволяет повысить масштабируемость и быстродействие системы на границе масштабируемости. Это достигается установкой на каждый SMP-узел двух серверов MySQL. Один из них связан с процессами IO, другой – с процессами Join [17]. Для сравнения архитектур «линейка» и «новая симметрия» проведена обработка конкатенации трех перестановок ПТ. Полученные результаты представлены на Рис. 4, a.

При использовании в кластере с архитектурой «линейка» 2-процессорных узлов с одним сервером MySQL один из процессоров остается практически незагруженным. Число исполнительных узлов «линейки» можно удвоить установкой на каждый рабочий узел кластера двух серверов MySQL и реализацией на каждом его процессоре пары (IO-JOIN) – т.н. конфигурация *«совмещенная линейка»* [18]. Она улучшает балансировку нагрузки между процессорами. На Рис. 4, b приведены полученные для нее графики *T(h)* (сплошная линия – сеть Gigabit Ethernet, пунктир – Infiniband) [18]. Сравнение графиков



a)



b)

Рис. 4. SUN-кластер, $V_{БД}$ = 5,4GB: a) Конкатенация 3-х перестановок ПТ; b) «Совмещенная линейка»

Рис. 3 и 4, b показывает, *что переход от «линейки» к «совмещенной линейке» дает рост эффективности в 2 раза*.

Значительное повышение эффективности консервативной СУБД на кластерной платформе, число узлов которой h кратно превышает h_G, достигается переходом к мультикластеризации (один мощный узел на каждый запрос) [17]. Мощный узел реализуется как монокластер, работающий на грани масштабируемости. На одном кластере реализуется множество одновременно функционирующих СУБД Clusterix с репликацией между ними консервативной базы данных. Для архитектур «линейка» либо «новая симметрия» на платформе SUN-кластера при $V_{БД}$ = 5,4GB наблюдался рост производительности по сравнению с однокластерной системой в 2 и 3 раза соответственно.

Эффекты самоорганизации. Кластерная СУБД – сложная диссипативная система [19]. В данном случае роль «трения» играют коллизии в сети, латентность, неравномерное распределение нагрузок по процессорам и др. Но существует и «отрицательное трение» – добавки в систему новых узлов. Они повышают производительность системы, пока рост противодействия не доминирует. Процессы в IO_r и $Join_r$ с ростом h усложняются.

Обратимся к исследованиям на SUN-кластере [20]. Исследовалась динамика «линейки» при $V_{БД}$ =5,4GB. Разработанная измерительная подсистема фиксировала объемы работ на отдельных этапах обработки. Усредненные графики изменения объемов работ на множестве 5 перестановок запросов ПТ, суммарных и по отдельным этапам, показаны на Рис. 5.

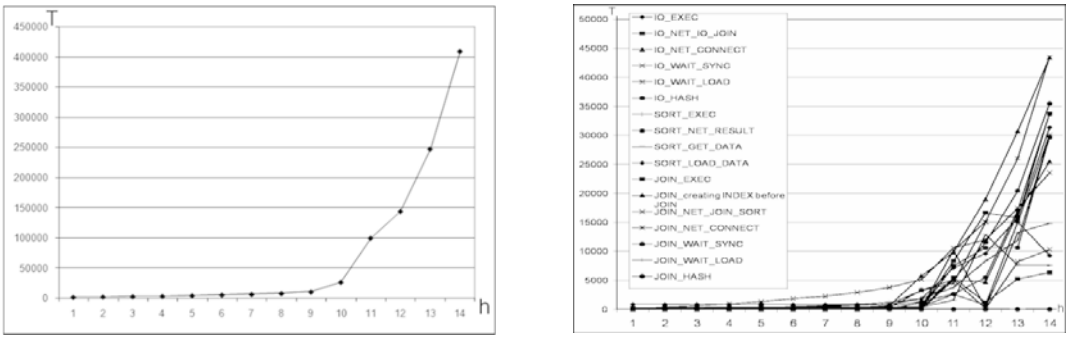
За объем работ было принято число эквивалентных скалярных операций длительностью 1 секунда каждая. При $h>10$ объем работ резко нарастает. Замечаем, что при $h > h_G$ высок удельный вес операций, так или иначе связанных с сетевыми передачами. По аналогии с влиянием «опосредованных сил трения» в физико-химических системах вводится

Постулат 3. *Резкое нарастание объемов работ при добавлении всего лишь 1 узла к уже имеющимся 10 означает переход кластера в точке бифуркации к новому режиму работы.*

2. Развитие системы Clusterix-N

Решаемая задача. Задачей данного раздела является анализ возможностей реализации экономичных консервативных СУБД повышенных объемов, сравнимых по эффективности 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. PostgreSQL является более совершенной открытой СУБД в сравнении с MySQL и активно позиционируется на территории России [7, 8]. Но MySQL позволяет использовать различные «движки» и имеет систему расширений [21]. Эти особенности упрощают и ускоряют разработку.

Согласно требованиям экономичности все исследовательские эксперименты были проведены на платформе GPU-кластера, состоящего из 7 узлов. Параметры узлов: 2 six-core E5-2640 CPU/2,5 GHz/DDR3 128GB; 2x GPU Tesla C-



a) b)

Рис. 5. Объемы работ на кластере: a) – суммарный; b) – по отдельным этапам

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 система – 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). За работу с SQL-запросами отвечало расширение Spark spark-sql.

Сравнительная оценка производительности Clusterix-подобных архитектур и Spark проводилась 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 $M(t_{зд}) = \left(\sum_{k=1}^{n} t_{зд}\right)/n$ и среднеквадратического отклонения $\sigma(t_{зд}) = \sqrt{M\left[t_{зд} - M(t_{зд})\right]^2}$, k – номер запроса ПТ, n – число запросов в ПТ. Поскольку значения M и σ немаловажны для пользователя, то сравнительная эффективность Clusterix-подобных систем определяется как кортеж $<\Delta T, \Delta M, \Delta\sigma>$, где $\Delta T = T_{Sp}/T_{Cl}$, $\Delta M = M_{Sp}/M_{Cl}$, $\Delta\sigma = \sigma_{Sp}/\sigma_{Cl}$ (Sp – Spark. Cl – Clusterix).

Типовая архитектура реляционной СУБД по Стоунбрейкеру и Хелерстейну [22] включает 5 компонентов: 1) менеджер коммуникации с клиентом; 2) менеджер управления процессами; 3) менеджер управления транзакциями; 4) общие компоненты и утилиты; 5) реляционный процессор. MySQL является самодостаточной СУБД и соответствует типовой архитектуре. Но все ее компоненты ориентированы на работу с одним потоком. Поэтому из MySQL использован только реляционный процессор. Компоненты (1-4) разрабатывались отдельно для каждой версии Clusterix-подобных СУБД. Заведомо неизвестно, как поведет себя система при ориентации на процессорные ядра и GPU-акселерацию в случае размещения БД в оперативной памяти процессоров I/O. Поэтому реализован полный процесс иерархического (***IS***) моделирования [4].

Постулат 4 [23]. *Решение поставленной задачи должна обеспечить эволюция Clusterix-подобных СУБД от начальной реализации принципов гибридной технологии (см. ниже принятое начальное состояние **IS-**модели).*

*Состояние **IS-**модели* – это совокупность взаимодействующих программных модулей. Их полная программная разработка названа *полным состоянием*. От пространства полных состояний можно перейти к *пространству параметров*. Под параметром понимается среднее время обработки одного запроса ПТ на том или ином этапе: *select-project, join, sort,* динамическая сегментация отношений, их индексация, сетевой и др. Для заданной платформы существует однозначное отображение пространства полных состояний в пространство параметров. По аналогии с принятым в синергетике [24], для каждого полного состояния выделяется *«параметр порядка»*, снижение влияния которого на производительность системы определяет переходы между состояниями-итерациями. В качестве такового принят временной параметр, имеющий максимальное значение для данного полного состояния [23].

В Табл. 2 показано изменение временных параметров системы на множестве рассмотренных итераций [25, 26]. Их формирование излагается далее в меру, принятой для обзора. Оно детально рассмотрено в [26-29]. Все разработки программной системы помещены в открытый доступ [26] и могут быть использованы заинтересованными организациями.

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

Табл. 2. Среднее время выполнения операций в Clusterix-N для итераций 1 – 5

	Итер. 1, сек	Итер. 2, сек	Итер. 3, сек	Итер. 4, сек	Итер. 5, сек
Передача данных	569,77	190,34	133,62	106,12	116,17
Удаление отношений R_i', $Rв_j$	375,27	308,17	180,03	140,91	41,06
Подготовка к загрузке	18,90	18,19	22,40	24,15	25,29
Хеширование данных	177,03	148,93	93,61	56,43	56,14
Загрузка данных в MySQL	466,64	562,10	447,28	164,01	85,18
Выполнение «join»	186,09	122,43	89,09	54,77	26,76
Выполнение «select-project»	522,73	685,37	159,86	55,40	59,93
Выполнение «sort»	162,61	164,00	178,42	42,90	43,41

времени и с ростом числа узлов могло приводить к сбоям в работе системы. В работе [25] показано, что повысить производительность можно переходом к архитектуре Clusterix-N (N – от New) путем разделения кластера на две различные части – блоки IO и JOIN – с независимой вариацией числа многоядерных узлов в каждом блоке. При этом число процессоров JOIN (физических или виртуальных) всегда не меньше числа процессоров IO.

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 работах [27, 30]), что позволило исключить динамическую сегментацию. Детальная программная разработка начального состояния [26] позволила создать своеобразные «модули-заготовки», которые в дальнейшем модифицировались для каждого нового состояния. Однако эффективность начального состояния оказалась существенно ниже, чем у Spark. Кроме того, уже при $V_{БД}$ < 100GB большой суммарный объем промежуточных отношений по некоторым запросам теста TPC-H приводил к перегрузке оперативной памяти узлов JOIN.

Первая итерация. Надежная работа с БД больших объемов требует использования стратегии «множество ядер в каждом блоке на одно отношение». Она восстановлена в первой итерации Clusterix-N [28, 29], но (в отличие от Clusterix) с передачей полученных в результате динамической сегментации промежуточных/временных отношений сегментов в целом. Хеширование осуществляет модуль HASH на выделенном узле с GPU-ускорителями, распределяя данные по всем процессорным ядрам уровня JOIN. Хеширование выполняется с использованием алгоритма деления (1). 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 (модуль управления), IO, JOIN, HASH, SORT. Как и ранее, БД распределена по узлам IO. Модули IO и JOIN реализуют стратегию «группа узлов на отношение». Архитектура этой итерации и ее алгоритмические особенности детально рассмотрены в работе [29]. Конфигурация кластера при проведении эксперимента такова: 2 узла IO, 3 узла JOIN, 1 узел HASH и 1 узел MGM, совмещающий модули MGM и SORT. Но и теперь Clusterix-N остается неконкурентоспособной. Результаты экспериментов для времени обработки ПТ: Clusterix-N – 19,7 час; Spark – 4,5 час. Они явно не в пользу Clusterix-N. Параметр порядка – временной вклад *сетевого уровня*.

Вторая итерация. Основная идея, положенная в ее основу, состоит в реализации операций динамического сегментирования промежуточных/временных отношений в модулях IO и JOIN с передачей хешированных данных напрямую между исполнительными узлами, минуя MGM (Рис. 6).

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 [28] и передаются сразу в определенные узлы JOIN. Модуль JOIN полностью повторяет

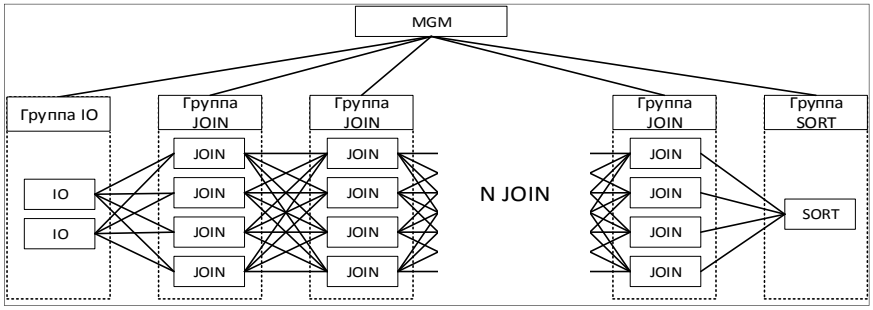


Рис. 6. Взаимодействие программных модулей Clusterix-N, итерация 2

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 MGM.

Экспериментальное исследование новой версии Clusterix-N производилось в конфигурации GPU-кластера (Рис. 7): 2 узла IO, 4 узла JOIN, 1 узел MGM. БД распределена по узлам IO. Время передач по сети уменьшилось ~ в 3 раза, операции *join* ускорились ~ в 1,5 раза. Но время выполнения ПТ составило 14,5 часа, т.е. говорить о возможной конкуренции со Spark по-прежнему не приходится. Параметр порядка для итерации 2 – время выполнения операций на уровне *select-project*.

Третья итерация. Для ускорения этих оп ераций надо уменьшить объем данных, обрабатываемых в одном узле, что требует увеличения числа узлов каждого блока. Нужного эффекта можно добиться возвратом к конфигурации «новая симметрия». Для этого требуется настройка 1) количества занимаемых процессорных ядер, 2) для связывания каждого модуля со «своим» GPU-ускорителем. Теперь для одного модуля вполне достаточно одного ускорителя. Новое распределение узлов GPU-кластера (Рис. 8): 6 узлов с модулями IO и JOIN, 1 узел MGM, совмещающий модули MGM и SORT.

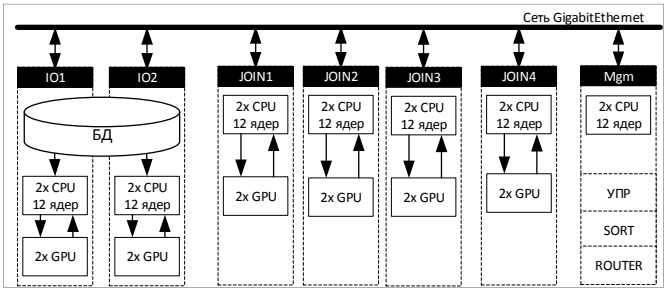


Рис. 7. Конфигурация программных модулей Clusterix-N, итерация 2

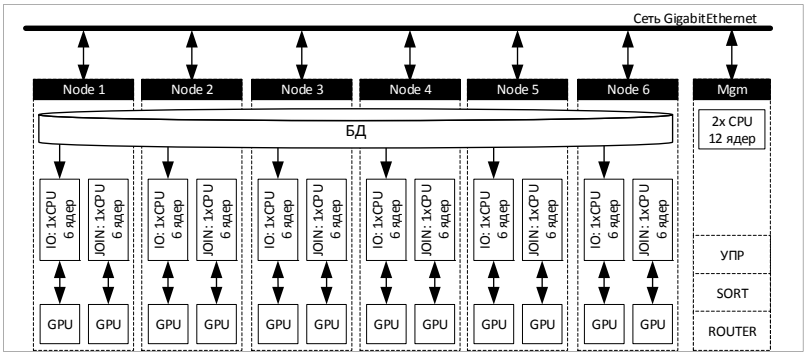


Рис. 8. Конфигурация программных модулей Clusterix-N в итерации 3

Каждому модулю выделен один CPU (6 ядер) и один GPU-ускоритель. На исполнительных узлах функционирует сразу две СУБД MySQL: одна – для IO, другая – для JOIN. Самой длительной операцией осталась *загрузка данных в MySQL* – «параметр порядка» для третьей итерации.

Четвертая итерация. Загрузка данных в MySQL для модулей JOIN ускорена увеличением числа ядер, на которых реализуются эти модули. В каждом узле итерации 3 процессор, на котором реализован модуль IO, простаивает продолжительное время. Возврат к архитектуре «совмещенная линейка» исключает эти простои со значительным ускорением загрузки. В каждом узле используется по одному GPU для хеширования результатов работы модулей IO и JOIN. Эффективность итерации 4 дополнительно повышена, во-первых, переходом в IO от хеширования БД по узлам к хешированию по ядрам. Это значительно ускорило *select-project*. Во-вторых, – переходом на более совершенную версию MySQL 8.0. Но к прежнему «параметру порядка» добавилось *удаление отработанных отношений* R_i', $Rв_j$ (Табл. 2).

Пятая итерация. В [21] сказано, что для движка *MEMORY* сервера MySQL блокировка производится на уровне таблиц**:** при выполнении операций изменения данных (*insert, update, delete, alter* и др.) запрещается доступ к таблице плоть до завершения названных операции. Детальный анализ исходных кодов помог установить, что блокировка одной таблицы вызывает блокировку всей памяти движка *MEMORY* в рамках одного процесса. Обойти это ограничение можно запуском нескольких экземпляров MySQL в количестве ядер CPU.

Работа со множеством MySQL требует налаживания множества связей Clusterix-N ↔ MySQL, контроля распределения данных результата хеширования и тиражирования запросов во множестве подключенных MySQL. В эксперименте для итерации 5 на каждом узле запускалось 13 серверов MySQL: 1 – для IO, 12 – для JOIN. MySQL для IO использует движок *InnoDB*, у которого нет указанных ранее проблем. Схема эксперимента показана на Рис. 9. Сравнение эффективностей итерации 5 и Spark дано в Табл. 3.

3. Возврат к идее мультикластеризации

При ориентации на использование сравнительно недорогих вычислительных кластеров в случае Big Data число исполнительных узлов кластера $h << h_G$. И все же понятие мощного узла можно ассоциировать с его многоядерностью даже при работе вдали от грани масштабируемости. При этом можно по иному (в сравнении с Clusterix-N) подойти к построению СУБД

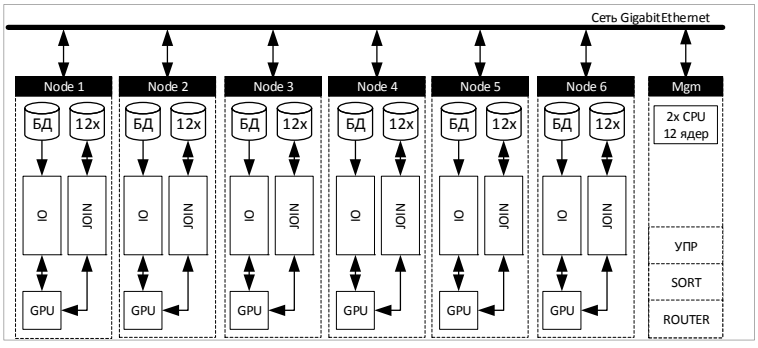


Рис. 9. Схема эксперимента

Табл. 3. Сравнение по T, M и σ итерации 5 и Spark, $V_{БД}$ = 120 GB

	Итерация 5 (и.5), мин.	Spark (S), мин.	Отношение и.5/S
T	276	261	1,06
M	4,8	3,1	1,54
σ	4,5	0,9	4,78

консервативного типа. Самостоятельное применение MySQL последних версий позволяет полностью использовать ресурсы многоядерных узлов (стратегия «ядро на запрос»). В работе [27] удалось обеспечить 100% загрузку всех ядер при специальной настройке MySQL 5.6 (технология *PerformSys* по терминологии [30]). Конфигурация экспериментального полигона для PerformSys на прежней платформе показана на Рис. 10. Распределение узлов GPU-кластера: 6 узлов исполнительных и 1 узел управляющий.

Управляющий узел выполняет функции роутеризации и ведения очереди запросов. Каждый исполнительный узел включает в себя собственно модуль сервера PerformSys и СУБД MySQL с полной копией БД на каждом узле. PerformSys использует стратегию «ядро на запрос». Поэтому в системе могут параллельно обрабатываться 72 запроса (6 узлов по 12 ядер в узле дают 72 ядра). Как только очередной запрос выполнен, на его место отправляется новый, вплоть до исчерпания очереди на управляющем узле. Требуется некоторая настройка сервера MySQL. Разработка программной системы PerformSys помещена в открытый доступ [31].

Сравнительные результаты выполнения ПТ на прежней платформе для двух БД объемом $V_{БД}$ = 60 и $V_{БД}$ = 120 GB представлены в Табл. 4. Превосходство PerformSys при $V_{БД}$ = 60 GB объясняется высоким параллелизмом работ: в системе сразу выполняется 72 запроса, в то время как в Clusterix-N всего 2. Но значения M и σ для Clusterix-N всегда меньше. Это немаловажно для пользователя. Низкая производительность PerfromSys при $V_{БД}$ = 120GB в основном объясняется недостаточным объемом оперативной памяти для хранения в инструментальной СУБД результата обработки и всех промежуточных/временных отношений по запросу.

Заключение

Нами показано, что использование регулярного плана обработки запросов и архитектурных решений согласно приведенным элементами теории при соответствующей программно-алгоритмической разработке экономичных консервативных СУБД класса BigData дает результаты, сравнимые по производительности с лучшими открытыми системами. Стоимость приобретения и ввода в эксплуатацию GPU-кластера, аналогичного использованному при проведении экспериментов, сравнительно невелика. Все версии программных систем Clusterix-N и PerformSys помещены в открытый доступ и могут быть использованы заинтересованными организациями.

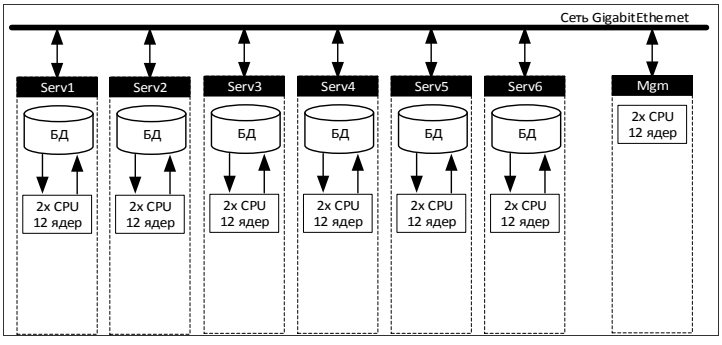


Рис. 10. Экспериментальный полигон для PerformSys

Табл. 4. Сравнительные результаты выполнения ПТ

$V_{БД}$, GB	60			120		
СУБД	PerformSys	Clusterix-N (итерация 4)	Spark	PerformSys	Clusterix-N (итерация 5)	Spark
T, мин	55,8	172,2	200	3015	276,13	260,6
M, мин	11,05	3,2	2,4	699,36	4,8	3,1
σ, мин	14,04	3,0	0,5	692,75	4,5	0,9

В работе над этим обзором использованы результаты, полученные в свое время по СУБД Clusterix кандидатами наук Е.В. Абрамовым и Р.Ш. Минязевым, магистрами Д.О. Шагеевым и А.В. Поповым. Авторы приносят им искреннюю благодарность за их труд.

Литература

1. Барсегян А.А., Куприянов М.С., Степаненко В.В., Холод И.И. Технологии анализа данных: Data Mining, Visual Mimning, Text Mining, OLAP // 2-е изд. – СПб.:БХВ-Петербург, 2007.
2. Cohen J., Dolan B., Dunlap M., Hellerstein J. M. and Welton C. MAD Skills: New Analysis Practices for Big Data // Proceedings of the VLDB Endowment Volume 2 Issue 2, August 2009. P. 1481-1492.
3. Raikhlin, V.A. Simulation of Distributed Database Machines //Programming and Computer Software. Vol. 22, Issue 2, 1996, P. 68-74.
4. Райхлин В.А. Конструктивное моделирование систем // Казань: Изд-во «Фэн» («Наука»), 2005.
5. EMC Education Services. Data Science and Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data //John Wiley & Sons.
6. Xin, Reynold & Rosen, Josh & Zaharia, Matei & J. Franklin, Michael & Shenker, Scott & Stoica, Ion. (2012). Spark: SQL and Rich Analytics at Scale // Proceedings of the ACM SIGMOD International Conference on Management of Data. 10.1145/2463676.2465288.
7. Российская отрасль СУБД продвигается на «слонах» //Connect. 2017. №5-6. С.34-38.
8. Российская СУБД Postgres Pro //Postgres Professional. 2018. URL:https://postgrespro.ru/products/postgrespro (дата обращения: 03.05.2018).
9. Codd E.F.. Providing OLAP to user-analysts: an it mandate // Technical Report. Apr. 1993.
10. Ульман Дж. Основы систем баз данных // М.: Финансы и статистка, 1983.
11. Райхлин В.А., Вершинин И.С., Классен Р.К., Гибадуллин Р.Ф., Пыстогов С.В.. Конструктивное моделирование процессов синтеза /Под ред. В.А. Райхлина // Казань: Изд-во «ФЭН» (Наука) АН РТ, 2020.
12. Воеводин В.В., Воеводин Вл.В.. Параллельные вычисления // СПб.: БХВ-Петербург, 2004.
13. Абрамов Е.В. Параллельная СУБД Clusterix. Разработка прототипа и его натурное исследование //Вестник КГТУ им. А.Н. Туполева. 2006. №2. С.50-55.
14. Райхлин В.А., Абрамов Е.В. Кластеры баз данных. Моделирование эволюции //Вестник КГТУ им. А.Н. Тупо лева. 2006. №3. С. 22-27.
15. Райхлин В.А., Абрамов Е.В., Шагеев Д.О. Эволюционное моделирование процесса выбора архитектуры кластеров баз данных //Высокопроизводит. паралл. вычисления на кластерных системах. Тр. 8 Межд. конф. HPC-2008. – Казань: Изд. КГТУ, 2008. С.249-256.
16. Райхлин В.А., Шагеев Д.О. Информационные кластеры как диссипативные системы //Нелинейный мир. Т.7. 2009. №5. С.323-334.
17. Райхлин В.А., Минязев Р.Ш. Мультикластеризация распределенных СУБД консервативного типа //Нелинейный мир. 2011. №8. С.473-481.
18. Райхлин В.А., Минязев Р.Ш. Анализ процессов в кластерах консервативных баз данных с позиций самоорганизации //Вестник КГТУ им. А.Н. Туполева. 2015. №2. С. 120-126.
19. Николис Г., Пригожин И. Познание сложного. – М.: УРС, 2003.
20. Минязев Р.Ш., Попов А.В. Временные доминанты кластеров баз данных //Методы моделирования /Под ред. В.А. Райхлина. Труды Респ. науч. семинара АН РТ «Методы моделирования». Вып.4. – Казань: Изд-во «Фэн» («Наука»), 2010. С.125-134.
21. 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).
22. 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.
23. Vadim A. Raikhlin, Roman K. Klassen. Clusterix-Like Big-Data DBMS //Data Science and Engineering, 5(1), p.80–93 (2020). DOI:10.1007/s41019-020-00116-2 http://link.springer.com/article/10.1007/s41019-020-0116-2
24. Haken, Hermann. Synergetics: Introduction and Advanced Topics // 2004, DOI: 10.1007/978-3-662-10184-1.
25. 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.
26. Классен Р.К. Clusterix-N. // BitBucket. 2019. URL: https://bitbucket.org/rozh/clusterixn/ (дата обращения: 09.03.2019).
27. Классен Р.К. Повышение эффективности параллельной СУБД консервативного типа на кластерной платформе с многоядерными узлами //Вестник КГТУ им. А.Н.Туполева. 2015. № 1. С. 112-118.
28. Классен Р.К. Ускорение операций хеширования с применением графических ускорителей //Вестник КГТУ им. А.Н.Туполева. 2018. № 1. С.134-141.
29. 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.
30. 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.
31. Классен Р.К. PerformSys // GitHub. 2019. URL: https://github.com/rozh1/PerformSys/ (дата обращения: 09.12.2018).

Райхлин Вадим Абрамович. Казанский национальный исследовательский технический университет им. А.Н. Туполева – КАИ, г. Казань. Профессор кафедры компьютерных систем, доктор физ.-мат. наук, профессор. Область научных интересов: конструктивное моделирование систем.E-mail: varaikhlin@gmail.com

Классен Роман Константинович. Казанский национальный исследовательский технический университет им. А.Н. Туполева – КАИ, г. Казань. Доцент кафедры компьютерных систем, канд. техн. наук. Область научных интересов: высокопроизводительные системы, big data, информационные технологии, интернет вещей. E-mail: klassen.rk@gmail.com

BIG DATA Class Conservative-Type Clusterix-Like DBMSs

V. A. Raikhlin, R. K. Klassen

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

Abstract. The reasonability of developing a conservative type DBMS with episodic data updating is determined by the features of OLAP-technologies. The issues of creating such DBMSs require serious discussion. In this review we systematize the main results of research of the research group of Clusterix KNITU-KAI on conservative DBMSs based on computational clusters. The purpose of the performed researches is actual: development of approaches to synthesize comparatively effective by the criterion “performance/cost” domestic Big Data class DBMSs. The comparison was made with the best foreign open systems. The developed DBMSs are available for use by organizations with limited financial resources. Due attention is paid to the elements of the theory of cluster DBMS of the conservative type. The basic configurations of Clusterix systems, the dynamics of such DBMSs, and the effects of their self-organization are considered. The research is based on the constructive system modeling methodology.
Keywords: conservative type cluster DBMSs, theory elements, basic configurations, process dynamics, self-organization effects, domestic Big Data class DBMSs, comparative efficiency.

DOI 10.14357/20718632240304 **EDN** DDMDGU

References

1. Barsegjan A.A., Kuprijanov M.S., Stepanenko V.V., Holod I.I. Data analysis technologies: Data Mining, Visual Mimning, Text Mining. OLAP. SPb.:BHV-Peterburg. 2007; 2 ed. In Russ.
2. Cohen J., Dolan B., Dunlap M., Hellerstein J. M. and Welton C. MAD Skills: New Analysis Practices for Big Data. Proceedings of the VLDB Endowment. 2009; 2 (2): 1481-1492.
3. Raikhlin, V.A. Simulation of Distributed Database Machines. Programming and Computer Software. 1996; 22 (2): 68-74.
4. Raikhlin V.A. Constructive modeling of systems. Kazan: Izd-vo «Fən» («Nauka»). 2005; In Russ.
5. EMC Education Services. Data Science and Big Data Analytics: Discovering, Analyzing, Visualizing and Presenting Data. John Wiley & Sons. 2015;
6. Xin, Reynold & Rosen, Josh & Zaharia, Matei & J. Franklin, Michael & Shenker, Scott & Stoica, Ion. Spark: SQL and Rich Analytics at Scale. Proceedings of the ACM SIGMOD International Conference on Management of Data. 2012; DOI: 10.1145/2463676.2465288.
7. Russian DBMS industry advances on "elephants". Connect. 2017; 5-6: 34-38. In Russ.
8. Russian DBMS Postgres Pro. Postgres Professional. 2018; Available from: https://postgrespro.ru/products/postgrespro [Accessed 03.05.2018]. In Russ.
9. Codd E.F.. Providing OLAP to user-analysts an it mandate. Technical Report. 1993;
10. Ullman, Jeffrey D. Principles of database systems. Galgotia publications. 1983.
11. Raikhlin V.A., Vershinin I.S., Klassen R.K., Gibadullin R.F., Pystogov S.V.. Constructive modeling of synthesis processes. Kazan: Izd-vo «Fən» («Nauka»). 2020; In Russ.
12. V.V. Voevodin, V.V. Voevodin. Parallel Computing. SPb.:BHV-Peterburg. 2004; In Russ.
13. Abramov E.V. Parallel DBMS Clusterix. Prototype development and its field study. Vestnik KGTU im. A.N. Tupoleva. 2006; 2: 50-55. In Russ.
14. Raikhlin V.A., Abramov E.V. Database Clusters. Modeling of evolution. Vestnik KGTU im. A.N. Tupoleva. 2006; 3: 22-27. In Russ.
15. Raikhlin V.A., Abramov E.V., Shageev D.O. Evolutionary modeling of the process of choosing the architecture of database clusters. 8 Mezhdunarodnaia Konferentciia "Vysokoproizvoditelnye parallelnye vychisleniia na

klasternykh sistemakh" HPC-2008. Kazan: Izd. KGTU. 2008: 249-256. In Russ.
16. Raikhlin V.A., Shageev D.O. Information clusters as dissipative systems. Nelineinyi mir. 2009; 7 (5): 323-334. In Russ.
17. Raikhlin V.A., Minyazev R.S. Multiclustering of distributed DBMS of conservative type. Nelineinyi mir. 2011; 8: 473-481. In Russ.
18. Raikhlin V.A., Minyazev R.Sh. Analysis of processes in clusters of conservative databases from the position of self-organization. Vestnik KGTU im. A.N. Tupoleva. 2015; 2: 120-126. In Russ.
19. Nicolis G., Prigogine I. Cognition of the complex. M.: URS. 2003; In Russ.
20. Minyazev R.Sh., Popov A.V. Temporal dominants of database clusters. Trudy Respublikanskogo nauchnogo seminara AN RT «Metody modelirovaniia». Kazan: Izd-vo «Fən» («Nauka»). 2010; 4: 125-134. In Russ.
21. Oracle. The MySQL Plugin API. MySQL Documentation. 2018; Available from: https://dev.mysql.com/doc/refman/5.7/en/plugin-api.html [Accessed: 09.04.2018]
22. Hellerstein J.M., Stonebraker M., Hamilton J. Architecture of a Database System. Foundations and Trends in Databases. 2007; 1 (2): 141-259.
23. Vadim A. Raikhlin, Roman K. Klassen. Clusterix-Like BigData DBMS. Data Science and Engineering. 2020; 5(1): 80–93. DOI:10.1007/s41019-020-00116-2
24. Haken, Hermann. Synergetics: Introduction and Advanced Topics. Springer. 2004; DOI: 10.1007/978-3-662-10184-1.
25. Raikhlin V.A., Klassen R.K. Comparatively inexpensive hybrid technologies of conservative DBMS of large volumes. Informatcionnye tekhnologii i vychislitelnye sistemy. 2018; 68(1): 46-59. In Russ.
26. Klassen R.K. Clusterix-N. 2019; Available from: https://bitbucket.org/rozh/clusterixn/ [Accessed: 09.03.2019]. In Russ.
27. Klassen R.K. Increasing the efficiency of a parallel DBMS of conservative type on a cluster platform with multicore nodes. Vestnik KGTU im. A.N.Tupoleva. 2015; 1: 112-118. In Russ.
28. Klassen R.K. Acceleration of hashing operations using graphics accelerators. Vestnik KGTU im. A.N.Tupoleva. 2018; 1: 134-141. In Russ.
29. Klassen R.K. Features of effective processing of SQL queries to databases of conservative type. Informatcionnye tekhnologii i vychislitelnye sistemy. 2018; 68 (4): 108-118. In Russ.
30. Klassen R.K. The program for regional load balancing to a conservative type database on the cluster platform «PerformSys». Certificate of state registration of the computer program No. 2017611785 of 09.02.2017. In Russ.
31. Klassen R.K. PerformSys. 2018; Available from: https://github.com/rozh1/PerformSys/ [Accessed: 09.12.2018]. In Russ.

Raikhlin Vadim A. 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 K. 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