DOI: 10.15514/ISPRAS-2020-32(1)-9

В ожидании нативных архитектур СУБД на основе энергонезависимой основной памяти

С.Д. Кузнецов, ORCID: 0000-0002-8257-028X <kuzloc@ispras.ru>
Институт системного программирования им. В.П. Иванникова РАН,
109004, Россия, Москва, ул. А. Солженицына, д. 25
Московский государственный университет имени М.В. Ломоносова,
119991, Россия, Москва, Ленинские горы, д. 1
Московский физико-технический институт,
141700, Россия, Московская область, г. Долгопрудный, Институтский пер., 9
НИУ «Высшая школа экономики»,
101978, Россия, Москва, ул. Мясницкая, д. 20
Российский экономический университет имени Г.В. Плеханова,
117997, Москва, Стремянный пер., 36

Аннотация. Многие специалисты в области управления данными считают, что появление доступной для практического использования энергонезависимой байт-адресуемой основной памяти (Non-Volatile Main Memory, NVM) приведет к появлению нового вида сверхскоростных систем управления базами данных (СУБД) с одноуровневым хранением данных (NVM-ориентированных СУБД). Однако, как отмечается во введении данной статьи, число исследователей, активно занимающихся исследованиями архитектур NVM-ориентированных СУБД, за последние годы не возросло. Наибольшую активность проявляют молодые исследователи, не боящиеся рисков, которые, безусловно, имеются в этой новой области. Во втором разделе статьи рассматривается состояние дел в области аппаратных средств NVM. Анализ показывает, что NVM в форм-факторе DIMM уже стали реальностью, и что в ближайшем будущем можно ожидать появления на рынке NVM-DIMM со скоростью обычной DRAM и стойкостью, близкой к стойкости жестких дисков. Третий раздел посвящен обзору родственных работ, среди которых наиболее продвинутыми представляются работы молодых исследователей Джоя Арулраджа и Исмаила Укида. В четвертом разделе мы утверждаем и обосновываем, что работы, выполненные к настоящему времени в области NVM-ориентированных СУБД, не привели к появлению чистой архитектуры. Этому мешает анализируемый нами набор ограничивающих факторов. В связи с этим, в пятом разделе мы приводим набросок «чистой» архитектуры NVM-ориентированной СУБД, на выбор которой влияют только стремление к простоте и эффективности. В заключение подводятся итоги статьи и утверждается потребность в дополнительном исследовании многих аспектов «чистой» архитектуры NVM-ориентированной СУБД.

Ключевые слова: энергонезависимая байт-адресуемая основная памяти; система управления базами данных; одноуровневое хранение данных; управление транзакциями; индексация; оптимизация запросов.

Для цитирования: Кузнецов С.Д В ожидании нативных архитектур СУБД на основе энергонезависимой основной памяти. Труды ИСП РАН, том 32, вып. 1, 2020 г., стр. 153-180. DOI: 10.15514/ISPRAS–2020–32(1)–9

In anticipation of native DBMS architectures based on non-volatile main memory

S.D. Kuznetsov, ORCID: 0000-0002-8257-028X <kuzloc@ispras.ru>
Ivannikov Institute for System Programming of the Russian Academy of Sciences,
25, Alexander Solzhenitsyn st., Moscow, 109004, Russia
Lomonosov Moscow State University,
GSP-1, Leninskie Gory, Moscow, 119991, Russia
Moscow Institute of Physics and Technology (State University),
9 Institutskiy per., Dolgoprudny, Moscow Region, 141700, Russia
National Research University, Higher School of Economics
20, Myasnitskaya Ulitsa, Moscow, 101978, Russia
Plekhanov Russian University of Economics,
36, Stremyanny lane, Moscow, 117997, Russia

Abstract. Many experts in the field of data management believe that the emergence of non-volatile byte-addressable main memory (NVM) available for practical use will lead to the development of a new type of ultra-high-speed database management systems (DBMS) with single-level data storage (native in-NVM DBMS). However, the number of researchers who are actively engaged in research of architectures of native in-NVM DBMS has not increased in recent years. The most active researchers are PhD students that are not afraid of the risks, which, of course, exist in this new area. The second section of the article discusses the state of the art in NVM hardware. The analysis shows that NVM in the DIMM form factor has already become a reality, and that in the near future we can expect the appearance on the market NVM-DIMMs with the speed of conventional DRAM and endurance close to that of hard drives. The third section is devoted to the review of related works, among which the works of young researchers are the most advanced. In the fourth section, we state and justify that the work performed so far in the field of in-NVM DBMS, did not lead to the emergence of a native architecture. This is hampered by the set of limiting factors analyzed by us. In this regard, in the fifth section, we present a sketch of the native architecture of an in-NVM DBMS, the choice of which is influenced only by the goals of simplicity and efficiency. In conclusion, we summarize the article and argues the need for additional research into many aspects of the native architecture of an in-NVM DBMS.

Keywords: non-volatile byte-addressable main memory, storage-class memory, database management system, single-level storage, transaction management, indexing, query optimization

For citation: Kuznetsov S.D. In anticipation of native DBMS architectures based on non-volatile main memory. Trudy ISP RAN/Proc. ISP RAS, vol. 32, issue 1, 2020. pp. 153-180 (in Russian). DOI: 10.15514/ISPRAS–2020–32(1)–9

1. Введение

Энергонезависимая основная память (Non-Volatile Main Memory, NVM, Persistent Main Memory, Storage-Class Memory) становится все более реальной. К массовому выпуску соответствующих чипов уже приступили гиганты электронной промышленности, такие как Intel и Samsung [1-3]. Расширяется набор технологий, используемых даже внутри одной компании (наиболее яркий пример представляет Intel [1, 3]) для производства чипов NVM.

Однако, как это ни странно, число исследователей, активно занимающихся исследованиями архитектур NVM-ориентированных СУБД, структур данных, методов и алгоритмов, обеспечивающих построение таких систем, за последние два-три года не увеличилось. На общем фоне выделяются два молодых исследователя: Джой Арулрадж (Joy Arulraj) и Исмаил Укид (Ismail Oukid). Первый из них получил степень PhD [4] в 2018 г. в университете Карнеги-Меллон, где он занимался исследованиями в группе баз данных вместе с Эндрю Павло (Andrew Pavlo), который был его руководителем в аспирантуре, и

в соавторстве, с которым сделаны основные публикации Арулраджа по тематике NVM-ориентированных СУБД (в частности, [5,6]). Стоит заметить, что Э. Павло, чрезвычайно заметный в настоящее время в сообществе баз данных, сам защитил диссертацию всего лишь в 2013 г. С 2018 г. Дж. Арулрадж работает в Технологическом институте Джорджии. В конце апреля 2019 г. ACM SIGMOD наградил Арулраджа премией им. Джима Грея за лучшую диссертационную работу; премия была вручена Джою на Международной конференции ACM SIGMOD по управлению данными в июле 2019 г. [7].

Исмаил Укид защитил диссертацию на степень PhD [8] в 2017 г. в Дрезденском техническом университете (Германия), где он учился и работал на кафедре баз данных (Дрезденская группа систем баз данных). Одновременно с этим он работал с компаниями Intel и SAP, и после защиты диссертации полностью перешел в SAP. Стоит заметить, что И. Усид сам пробился на крупнейшие конференции в области баз данных и в самые рейтинговые журналы (например, [9, 10]). Заслуживает внимания также тот интересный факт, что И. Укид (как и Дж. Арулрадж) вообще не занимался исследованиями в области баз данных до начала работы над диссертацией.

Чем же можно объяснить это сравнительно равнодушное отношение «взрослых» представителей сообщества баз данных к актуальной и перспективной теме NVM-ориентированных СУБД? Ответить на этот вопрос можно по-разному.

Во-первых, данные о реальных характеристиках промышленных образцов MVM недостаточны и противоречивы. Производители соответствующих аппаратных средств недостаточно информируют открытым образом научную общественность об их характеристиках, а достоверность цифр, публикуемых популярных Интернет-изданиях, невозможно проверить.

Во-вторых, видимо, из сиюминутных маркетинговых соображений, вендоры NVM на первых порах предпочитают предлагать продукты, основанные на технологии NVM, в форм-факторе блочных устройств внешней памяти, а не чипов энергонезависимой байт-адресуемой основной памяти. (Текущему состоянию аппаратных средств NVM посвящается следующий раздел статьи.)

В-третьих, известно, что на выбор направлений научных исследований (по крайней мере, в области компьютерных наук) в значительной степени влияет возможность получения финансирования (чаще всего в виде грантов). Во всем мире фонды, поддерживающие научные исследования, требуют гарантий получения при выполнении проекта практических результатов, близких к возможности реального использования. В настоящее время относительно проектов в направлении NVM-ориентированных СУБД таких гарантий никто дать не может.

Наконец, следует заметить, что заслуженные члены сообщества баз данных на самом деле в своих исследованиях затрагивают вопросы, прямо относящиеся к архитектуре и алгоритмам NVM-ориентированных СУБД. Формально эти исследования относятся к области СУБД с хранением баз данных в обычной основной памяти (in-memory СУБД), но при этом исследователи абстрагируются от того обстоятельства, что содержимое традиционной основной памяти утрачивается при отключении питания.

Во втором разделе статьи мы кратко обсудим состояние дел в области аппаратных средств NVM. Краткий обзор работ, прямо или косвенно связанных с архитектурой, структурами данных и алгоритмами NVM-ориентированных СУБД приводится в разд. 3.

По мнению автора этой статьи, все публикации, посвященные NVM-ориентированным СУБД, обладают одним общим свойством, которое можно считать достоинством или недостатком: их авторы не решаются замахнуться на полноценную разработку СУБД с одноуровневой системой хранения, когда базы данных целиком и полностью хранятся в энергонезависимой основной памяти, и никакая внешняя блочная память вообще не

используется. Причины того, что «чистую» архитектуру NVM-ориентированной СУБД до сих пор никому предложить не удалось, рассматриваются в разд. 4.

С одной стороны, расчет на использование иерархически организованной системы хранения с включением NVM в слой между энергозависимой основной памятью и блочной флэш-памятью (solid-state disks, SSD) ближе к сегодняшней реальности. С другой стороны, именно полный переход к одноуровневой энергонезависимой системе хранения может позволить упростить организацию СУБД и значительно увеличить их эффективность. В разд. 5 приводится набросок «чистой» архитектуры такой СУБД.

2. Аппаратные средства NVM

За прошедшие пару лет основные технологические тренды NVM практически не изменились. Как и раньше (см., например, [11]), основные компании, заинтересованные в производстве NVM, делают ставки на методы энергонезависимого хранения данных с побайтовой адресацией на основе физических эффектов

- фазовых переходов (Phase-Change Memory, PCM),
- изменения сопротивления диэлектриков (Resistive Random-Access Memory, ReRAM) и
- переноса спинового момента (Spin-Torque-Transfer Memory, STT-RAM).

Последняя в этом списке технология относится к более общему классу магнитно-резистивных решений (magnetoresistive random-access memory, MRAM). В конечном счете, все разновидности MRAM, как и ReRAM, основаны на изменении сопротивления; в случае MRAM сопротивление изменяется при изменении ориентации намагниченности.

В табл. 1 приведены примерные характеристики NVM разновидностей PCM, ReRAM и STT-RAM. К сожалению, в открытом доступе не удается найти значения таких характеристик от производителей и/или разработчиков NVM. В таблице собраны неофициальные показатели, найденные в различных публикациях ([3, 5, 12, 13]) за период с 2016 по 2019 гг. Местами они довольно сильно различаются, но общий вывод сделать можно: на сегодняшний день ближе всех к DRAM по скоростным характеристикам находится STT-RAM.

Табл. 1. Приблизительные характеристики разных видов NVM
Table 1. Approximate characteristics of different types of NVM

	Источник	Год публикации	PCM	STT-RAM	ReRAM	DRAM
Время чтения (нс)	[3]	2019	20-70	10-30	10	10-50
	[10]	2018	250	20	100?	100
	[5]	2017	50	20	100	60
	[13]	2016	50	10	10	50
Время записи (нс)	[3]	2019	50-500	13-95	1-100	10-50
	[10]	2018	250	20	100?	100
	[5]	2017	150	20	100	60
	[13]	2016	500	50	50	50
Стойкость	[3] *	2019	$1\text{-}10^8$	10^{15}	10^{10}-10^{12}	$> 10^{17}$
	[10] DWPD **	2018	100	↑	40?	↑
	[5] *	2017	10^{10}	10^{15}	10^8	$> 10^{16}$
	[13] *	2016	10^8-10^9	$> 10^{15}$	10^{11}	$> 10^{15}$
* Число циклов записи						
** DWPD, Drive Writes Per Day, допустимый объем суточной записи относительно емкости самого устройства						

Немного смущает указываемое одним из авторов различное время чтения и записи у STT-RAM. Нигде в литературе не удается найти какие-либо сведения об обязательности наличия этого явления. Поскольку двое других авторов указывают для STT-RAM

одинаковое время чтения и записи (и практически такое же, как у DRAM), далее в этой статье будем считать, что так оно и есть. Скорее всего, ближайшее будущее покажет, насколько оправдан этот оптимизм.

С разработкой разных видов NVM связан ряд компаний. В частности, лидером в области технологии NVM на основе фазовых переходов принято считать компанию Micron Technology [14]. Технологией ReRAM наиболее активно занимаются компании Panasonic [15] и Crossbar [16]. Что касается разработки STT-RAM, можно выделить компании Everspin Technologies [17] и Crocus Technology [18] (и ее российскую «дочку» Крокус Наноэлектроника [19]). Однако, несмотря на значительный вклад этих компаний в развитие технологий NVM, ни одна из них не является вендором мирового уровня, способным (или хотя бы желающим) производить доступные для широкого использования модули энергонезависимой памяти.

Реальным прорывом в этом направлении является начало массового производства продуктов категории NVM компаниями Intel и Samsung [1]. Еще в 2015 г. компании Intel и Micron объявили о создании технологии 3D XPoint – NVM на основе PCM (для разработки технологии NVM на основе фазовых переходов в 2006 г. компании Intel и Micron создали специальное совместное предприятие IM Flash Technologies; 3D XPoint – не первая, но самая успешная разработка IM Flash Technologies).

В марте 2017 г. Intel объявила о промышленном производстве модулей 3D XPoint под брендом Optane. По маркетинговым соображениям (а также, по-видимому, из-за того, что по своим скоростным характеристикам NVM Optane занимает нишу между DRAM и флэш-памятью) на первых порах для продвижения Optane Intel использовала два решения: кэш внутри традиционных дисковых устройств и твердотельные диски целиком на основе Optane. Оба решения оказались весьма эффектными, и о SSD на основе Optane мы еще поговорим ниже в этом разделе.

В августе 2018 г. компания Intel объявила о начале производства DIMM на основе Optane. С одной стороны, DIMM является общепринятым форм-фактором современных энергозависимых модулей основной памяти. С другой стороны (увы!), DIMM на основе Optane невозможно использовать вместе с обычными процессорами Intel. Одновременно с объявлением DIMM на основе Optane Intel анонсировала новую линейку процессоров Cascade Lake AP (Advanced Processor) Xeon CPU, в которых эта возможность будет обеспечена. Так что использовать Intel Optane NVM в качестве замены RAM пока проблематично.

Со своей стороны, компания Samsung в начале 2018 г. объявила о начале производства SSD на основе собственной технологии Z-NAND (многослойная NAND, или 3D NAND). Мы не беремся судить об особенностях этой технологии, но утверждается, что она позволила повысить скорость чтения из внешней памяти чуть ли не в 10 раз по сравнению с обычными SSD на основе флэш-памяти. В [1] приводятся следующие характеристики SSD на основе Z-NAND от Samsung и SSD на основе Optane от Intel. При емкости SSD в 800 Гб Samsung SZ985 Z-NAND поддерживает в секунду 750,000/170,000 произвольных блочных обменов с внешней памяти по чтению или записи соответственно. При той же емкости Intel Optane P4800X поддерживает в секунду 550,000/500,000 таких обменов. Вполне конкурентоспособные результаты, несмотря на принципиальное различие применяемых технологий.

Тему быстрых устройств внешней памяти и связанную с их появлением потребность в пересмотре архитектуры и алгоритмов организации СУБД мы кратко затрагивали в [11]. Понятно, что эта потребность только усиливается в связи с новыми достижениями Intel и Samsung, но в данной статье это обсуждаться не будет. Основные же события, которые к этой статье имеют непосредственное отношение, произошли в 2019 г.

На конференции International Solid-State Circuits в феврале 2019 г. разработчики из компании Intel заявили о готовности компании начать промышленный выпуск модулей энергонезависимой памяти STT-MRAM [20]. Интересно, что на той же конференции в выступлении другой группы разработчиков было заявлено о готовности Intel к разработке моделей памяти и на основе технологии ReRAM [21]. Intel планирует применять эту более дешевую и менее быструю память в областях Интернета вещей и автомобилестроения. Хотя ко времени написания этой статьи официальных заявлений компании Intel, анонсирующих соответствующие продукты, еще не было, думается, что указанные выступления на конференции могли состояться только с полного благословения руководства компании. Так что совсем скоро мы сможем получить от Intel высокопроизводительные модули памяти STT-MRAM.

А может быть, и не от Intel, потому что буквально через две недели после выступления разработчиков из Intel компания Samsung официально объявила [2] о начале поставок модулей энергонезависимой памяти на основе своей технологии eMRAM на основе STT. Пока говорится, что модули eMRAM будут использоваться в микроконтроллерах, устройствах Интернета вещей и почему-то в системах искусственного интеллекта, но если технология успешно себя зарекомендует, то Samsung сможет обеспечит и выпуск моделей памяти в формате DIMM.

Общим выводом этого раздела является то, что высокопроизводительная энергонезависимая основная память стала реальностью, и работа в области NVM-ориентированных СУБД приобрела еще большую актуальность.

3. Родственные работы

Как отмечалось во введении, в мире проводится недостаточно активная исследовательская работа по поиску архитектур, структур данных, алгоритмов и методов, необходимых и достаточных для построения полноценных NVM-ориентированных СУБД. Несмотря на это, число публикаций, относящихся к специфике управления данными в энергонезависимой основной памяти, достаточно велико. Однако в подавляющем большинстве этих публикаций рассматриваются частные вопросы, в отрыве от общей задачи построения СУБД нового поколения.

Возможно, для полноты картины стоило бы написать обзор всех доступных работ, так или иначе касающихся использования NVM в системах управления данными. Но это выходит за пределы текущих интересов автора. В этой статье нас интересуют архитектурные и алгоритмические черты NVM-ориентированных СУБД. Работ, посвященных решению задачи именно в этой постановке, удивительно мало. Именно их обзору посвящен этот раздел.

3.1 Использование NVM в существующих СУБД

Стратис Виглас (Stratis D. Viglas) из Эдинбургского университета в своей краткой статье 2015 г. [22] опирается на предположение о том, что доступная энергонезависимая основная память обеспечивает скорость доступа, близкую к возможностям современной DRAM, но при этом ассиметричная: операции чтения выполняются быстрее операций записи. Поэтому, по мнению автора, NVM не сможет заменить ни DRAM, ни HDD. Предлагаются два пути интеграции NVM в иерархию сред хранения данных: путем использования в качестве устройства внешней памяти и на основе подключения таким же способом, что и основная память. Автор отмечает недостаточную исследованность области управления данными с использованием NVM и призывает активизировать работу в этом направлении.

Навид Уль Мустафа (Naveed Ul Mustafa) и др. из университета Билькент (Анкара, Турция) и Барселонского суперкомпьютерного центра в [23] описывают свою работу по замене

подсистемы хранения данных в СУБД PostgreSQL, изначально ориентированной на использование HDD, на разработанную ими подсистему, ориентированную на применение NVM. Для обеспечения доступа к NVM авторы использовали файловую систему PMFS (Persistent Memory File System)[1], которая отображает файлы, хранящиеся в энергонезависимой основной памяти, в виртуальную память работающего с ними процесса. Тем самым, хотя обеспечивался «прямой» доступ к NVM без потребности переписи данных из NVM в RAM и обратно, на уровне файловой системы поддерживается блочная структура NVM. Кроме изменений подсистемы хранения данных, в PostgreSQL были ликвидированы за ненадобностью кэш в основной энергозависимой памяти и журнал REDO.

Авторы сравнивали производительность исходного варианта PostgreSQL с использованием SSD и измененного варианта системы с использованием эмулируемой NVM на основе бенчмарка TPC-H (моделирующего рабочую нагрузку, свойственную системам поддержки принятия решений). Результаты не слишком вдохновляют: время выполнения запросов сократилось в среднем всего на 20%. По-видимому, это связано с тем, что фактически сохраняется иерархическая система хранения данных, и для доступа к очередной порции данных из базы данных системе всегда требуется обращение к файловой системе.

В [24] Михня Андрей (Mihnea Andrei) и др. из компании SAP SE и немецкого подразделения компании Intel представляют первую попытку внедрить использование NVM в СУБД HANA. HANA – это заново разработанная в SAP производственная СУБД с поколоночным хранением таблиц в основной памяти (in-memory columnar DBMS). HANA поддерживает как транзакционную, так и аналитическую рабочую нагрузку, причем в приложениях SAP даже в транзакционной рабочей нагрузке более 80% операций составляет чтение данных, так что поколоночное хранение таблиц в основной памяти является оправданным.

В HANA поддерживается высокий уровень доступности и надежности данных на основе резервного копирования, репликации и журнализации в режиме REDO. Для всего этого используется дисковая внешняя память. Для полного перехода на NVM потребовалась бы серьезная переделка системы, и в своей первой попытки внедрения NVM разработчики SAP на это не решились. По техническим соображениям в NVM, используемой в форм-факторе DIMM, размещаются лишь редко изменяемые крупные фрагменты описателей таблиц.

В статье не приводятся какие-либо убедительные показатели преимуществ полученного варианта системы, но отмечается, что, во-первых, общая организация HANA хорошо подходит для перехода к полноценному использованию NVM. Во-вторых, для этого предстоит решить массу технических задач. В-третьих, пути к будущему переводу HANA на NVM намечены в статьях Исмаила Укида и др. (авторы, в число которых сам Укид не входит, ссылаются на статьи [30, 31, 33]). На работах Укида мы остановимся в конце этого раздела.

Александр ван Ренен и др. из Мюнхенского технического университета (Германия) и компании Fujitsu в [25] описывают свою разработку подсистемы хранения данных с использованием NVM, которую они оправдывают следующей цитатой из недавнего интервью Майкла Стоунбрейкера (Michael Stonebraker)[2]: модели памяти NVM «недостаточно быстры, чтобы заменить основную память, и они недостаточно дешевы, чтобы заменить диски, и они недостаточно дешевы, чтобы заменить флэш-память». Поскольку в интервью Стоунбрейкер отвечает на вопрос о 3D XPoint или ReRAM, с ним нельзя не согласиться, но, что бы он сказал о технологии STT-RAM 2019 года?

[1] https://github.com/linux-pmfs/pmfs

[2] https://www.nextplatform.com/2017/08/15/hardware-drives-shape-databases-come/

Тем не менее, следуя этому указанию, авторы статьи предлагают архитектуру системы хранения данных, в иерархии которой NVM занимает место между DRAM и SSD. Утверждается, что в результате ряда технических приемов, описываемых в статье, авторам удалось получить решение, конкурентоспособное по отношению к in-memory СУБД и СУБД, использующим NVM в качестве единственного уровня хранения данными. Возможно, подход авторов действительно соответствует соотношению скоростей DRAM, ReRAM и SSD, имевшихся в 2018 г. Будет ли он хорош, например, при наличии быстрых SSD на основе Intel Optane или Samsung Z-NAND, совершенно непонятно.

Заканчивают авторы статью цитатой из книги еще одного классика баз данных покойного Джима Грея[3], которой мы тоже руководствуемся в своей работе: «Не позволяйте себя дурачить книгам о сложности или всяким сложным и малопонятным алгоритмам, которые вы найдете в этой книге или где-то еще. Хотя нет учебников по простоте, простые системы работают, а сложные нет».

На конец обзора мы оставили работы Джоя Арулраджа и Исмаила Укида, которые, по нашему мнению, в настоящее время больше других исследователей продвинулись в разработке архитектур, алгоритмов и методов, которые пригодны для построения NVM-ориентированных СУБД.

3.2 Использование NVM в СУБД Peloton

Джой Арулрадж выполнил все работы, которые составили его диссертацию, в рамках проекта Peloton[4], лидером которого является молодой и чрезвычайно энергичный исследователь и преподаватель Энди Павло (он же был научным руководителем Джоя). В проекте участвуют как представители старшего поколения исследователей университета Карнеги-Меллон (Тодд Моури (Todd C. Mowry) и Энтони Томашич (Anthony Tomasic)), так и около 20 студентов и аспирантов, в число которых до защиты диссертации входил и Джой Арулрадж. Как уже отмечалось, в настоящее время Джой работает доцентом в Технологическом институте Джорджии, но во всех его публикациях, кроме [7], он аффилирован с университетом Карнеги-Меллон.

Peloton – это новая «самоуправляемая» in-memory СУБД, в которой все аспекты поведения системы контролируются интегрированным планировщиком, который не только оптимизирует систему для выполнения текущей рабочей нагрузки, но и прогнозирует будущую рабочую нагрузку, чтобы система к ней подготовиться. Для этого в систему интегрированы черты искусственного интеллекта, в частности, машинного обучения.

Но кроме этого, в Peloton реализована поддержка энергонезависимой основной памяти. Судя по публикациям, это сделано, главным образом, стараниями Джоя Арулраджа при содействии и под руководством Энди Павло. Результаты, полученные Арулраджем, опубликованы в многочисленных статьях [5-6, 26-29], но окончательным текстом, в котором эти результаты четко изложены и упорядочены, является его диссертация [4], которой мы и будем следовать в своем обзоре.

Как отмечает автор, в его диссертации представлены разработка и реализация СУБД Peloton, ориентированной на использование NVM. Полученная архитектура системы обладает несколькими важными преимуществами перед другими существующими системами, из которых в данной статье нам наиболее интересны следующие:

- в архитектуре подсистемы хранения данных используются свойства долговечности и байтовой адресации NVM; обеспечивается экономное использование

[3] Jim Gray and Andreas Reuter. Transaction Processing: Concepts and Techniques. Morgan Kaufmann, 1993

[4] https://pelotondb.io/

энергонезависимой памяти и увеличивается срок ее эксплуатации за счет уменьшения числа записей;

- в ней применяется новый протокол журнализации и восстановления «журнализация с отложенной записью в журнал» (Write-Behind Logging, WBL), позволяющий обеспечить высокий уровень доступности;

Обсудим эти особенности архитектуры более подробно.

3.2.1 Подсистемы хранения данных

Для случая, когда для хранения данных используется только NVM, а внешняя память вовсе не используется, были разработаны и оптимизированы три разных подсистемы хранения данных:

- подсистема с обновлением данных прямо на текущем месте их хранения (In-Place Updates Engine, NVM-InP);
- подсистема с созданием новой копии данных при их обновлении (Copy-on-Write Updates Engine, NVM-CoW) и
- подсистема с журнально-структурированными обновлениями (Log-structured Updates Engine, NVM-Log).

Если не вдаваться в технические детали, при разработке подсистемы NVM-InP применялись достаточно традиционные методы с учетом специфики NVM – на основе журнализации с упреждающей записью изменений (Write-Ahead Log, WAL). Основным отличием от случая, когда данные хранятся во внешней памяти, является отсутствие потребности в повторном выполнении операций при перезапуске системы после сбоя, поскольку результаты любой зафиксированной транзакции гарантированно находятся в долговременной NVM. Журнал поддерживается в виде списка в NVM, и «верхняя» часть списка, соответствующая уже зафиксированным транзакциям, освобождается.

Поэтому при выполнении операции INSERT в журнал записывается только указатель на заново вставляемую строку, позволяющий удалить ее при откате транзакции. При выполнении операции DELETE в журнал тоже записывается только указатель за удаляемую строку; память, занимаемая этой строкой, освобождается только после фиксации соответствующей транзакции, что позволяет при откате транзакции восстановить все индексы на основе сохраненного в журнале указателя. Понятно, что при выполнении операции UPDATE приходится сохранять в журнале предыдущие значения изменяемых полей строки.

Подсистема NVM-CoW, фактически, является системой хранения с поддержкой не более двух копий всех элементов данных, включая строки таблиц и узлы B+-деревьев (индексов). Для каждой таблицы и каждого B+-дерева в NVM поддерживаются два справочника – текущий (current), сохраняющий ссылки на копии строк таблиц или узлов B+-дерева, созданные зафиксированными транзакциями, и измененный (dirty), который содержит ссылки на соответствующие объекты, созданные незафиксированными транзакциями.

При фиксации транзакции все измененные справочники, содержащие ссылки на копии объектов, которые были созданы при выполнении данной транзакции, делаются текущими. Для этого насильственно выталкиваются в NVM все кэши процессора, и справочник объявляется текущим. Для этого для каждого справочника в NVM поддерживается специальная «мастер-запись», содержимое которой показывает, какой справочник является текущим. Мастер-запись изменяется атомарной командой записи. Кстати, для экономии операций выталкивания кэша и обновления мастер-записей в [4] транзакции фиксируются не по одной, а пакетами.

Понятно, что при таком управлении хранением данных при перезапуске СУБД после сбоя не требуется откатывать какие-либо транзакции, а значит, не требуется и поддерживать журнал UNDO в NVM. Для отката транзакции при наличии работоспособной системы достаточно запоминать в локальной памяти транзакции список координат в измененной справочнике копий объектов, созданных данной транзакцией.

Заметим еще по поводу NVM-CoW, что Арулрадж проводит аналогию между используемым подходом и «теневым» механизмом, который использовался в System R для установки контрольных точек базы данных [30]. По нашему мнению, этот подход в большей степени напоминает двухверсионный двухфазный метод синхронизационных блокировок (Two Version 2PL, 2V2PL), описанный в пятой главе [31]. Поддержку двух копий на уровне строк таблицы трудно представить без соответствующей поддержки изолированности транзакций: если некоторая транзакция изменяет некоторую строку таблицы, то до конца этой транзакции никакая другая транзакция не должна иметь возможности изменять эту строку, но должна иметь возможность прочитать копию этой строки, указатель на которую содержится в текущем справочнике. К сожалению, в [4] и других публикациях Арулраджа ничего не говорится об этом аспекте NVM-CoW, из-за чего реализация этой подсистемы остается непонятной.

Третья подсистема NVM-Log основана на использовании LSM-деревьев (Log-Structured Merge-tree), введенных в 1996 г. Патриком О'Нейлом (Patrick O'Neil) и др.[5] Вся база данных поддерживается в виде последовательности специальных хранилищ – MemTable. Изменения (строка таблицы при выполнении операции INSERT, новое содержимое изменяемых столбцов строки при выполнении UPDATE, пометка об удалении строки для DELETE) заносятся в текущую изменяемую MemTable. Когда ее заполнение достигает установленного порога, создается новая текущая MemTable, а предыдущая становится неизменяемой. Все MemTable хранятся в NVM, неизменяемые – сливаются. Для доступа к базе данных поддерживаются индексы – B+-деревья и фильтры Блюма. Для восстановления базы данных поддерживается журнал UNDO WAL, в котором сохраняются не значения, а только ссылки с места их хранения в MemTable.

Для проведения экспериментов была реализована специальная «легковесная» СУБД с использованием эмулятора NVM. Для сравнения характеристик подсистем применялись тестовые наборы YCSB (Yahoo! Cloud Serving Benchmark)[6] и TPC-C[7]. Основным выводом является то, что для рабочих нагрузок с большим числом изменений лучше работает NVM-InP, и для рабочих нагрузок с преобладание операций чтения – NVM-CoW. Эти результаты согласуются с интуицией.

3.2.2 Откладываемая запись в журнад

Исторически в СУБД, использующих журнализацию, применяются протоколы журнализации с упреждающей записью в журнал (Write Ahead Log). В СУБД категории in-memory, к которым относится Peloton, восстановление базы данных происходит путем загрузки в основную память из внешней памяти копии базы данных, соответствующей последней контрольной точке, с последующим откатом изменений, выполненных транзакциями, которые не завершились к моменту сбоя (UNDO), и повторным применением операций, выполненных транзакциями, которые не зафиксировались до установки последней контрольной точки, но успели зафиксироваться до точки сбоя (REDO). Если для управления транзакциями используется какая-либо разновидность

[5] The Log-Structured Merge-Tree (LSM-Tree). Acta Informatica, vol. 33, issue 4, 1996, pp. 351-385.
[6] https://en.wikipedia.org/wiki/YCSB.
[7] http://www.tpc.org/tpcc/.

многоверсионного протокола (Multi-Version Concurrency Control, MVCC)[8], то можно обойтись без UNDO, но REDO требуется, а значит, в журнал в долговременной памяти требуется помещать записи по поводу каждой операции изменения базы данных.

Придуманный Арулраджем протокол журнализации с откладываемой записью в журнал (Write-Behind Logging) позволяет значительно сократить расходы на журнализацию и ускорить восстановление базы данных после сбоев. Протокол рассчитан на использование в СУБД, использующих для хранения баз данных только NVM, но применяющих для служебных целей и обычную энергозависимую основную память (DRAM). На рабочей фазе выполнения транзакции результаты всех операций изменения базы данных сохраняются в DRAM в таблице измененных кортежей (Dirty Tuple Table, DTT). Эта таблица применяется для индивидуальных откатов транзакции.

При выполнении операции COMMIT (а фиксация выполняется сразу для группы транзакций) сначала все изменения каждой фиксируемой транзакции записываются в постоянное место хранения таблиц в NVM, а затем в журнал пишется запись, содержащая временную метку фиксации транзакции из данной группы, которая последней закончила запись в NVM элементов своей DTT, а также временную метку со значением, заведомо большим временной метки фиксации транзакций из следующей группы.

После сбоя в NVM заведомо содержатся все результаты зафиксированных транзакций, но, возможно, еще и некоторые результаты транзакций, не зафиксированных до момента сбоя. При использовании любого варианта MVCC эти результаты не были до сбоя видны другим транзакциям, и работу системы можно возобновить немедленно, одновременно запустив служебный процесс сборки мусора.

3.3 Исследования в Дрезденской группе систем баз данных

Дрезденская группа систем баз данных (Database Systems Group Dresden) сформировалась на основе кафедры баз данных Института системных архитектур факультета компьютерных наук Дрезденского технического университета[9]. С момента основания кафедры ее заведующим и лидером группы систем баз данных является профессор Вольфганг Лехнер (Wolfgang Lehner). Он защитил диссертацию PhD в 1998 г. в Университете Эрлангена-Нюрнберга им. Фридриха-Александра, работал в США в компании IBM, а с 2002 г. работает в Дрезденском техническом университете.

В Дрезденской группе баз данных выполняется много проектов с участием как штатных сотрудников, так и студентов, но судя по всему, напрямую с тематикой NVM-ориентированных СУБД была связана только работа Исмаила Укида, которую он выполнял во время своего пребывания в аспирантуре с 2013 по 2017 гг., где его научным руководителем являлся Лехнер. Под его же руководством Укид подготовил диссертационную работу, которую защитил в самом конце 2017 г. При выполнении своего диссертационного проекта Укид, по всей видимости, пользовался инфраструктурой группы, а также сотрудничал с исследователями компаний SAP и Intel. В частности, для проведения экспериментов на основе инфраструктуры Дрезденской группы и при активном участии студентов был реализован прототип NVM-ориентированной СУБД SOFORT [32].

После защиты диссертации Исмаил Укид сразу перешел на работу в компанию SAP (скорее всего, в группу, занимающуюся разработкой СУБД HANA). Информация о его деятельности с новом качестве пока недоступна, но за время учебы и работы в Дрезденской группе он опубликовал свои результаты в ряде интересных статей ([9-10, 32-

[8] В [4] (а также в [5, 6]) говорится, что в Peloton используется MVCC, но нигде не уточняется, какой вариант.

[9] https://wwwdb.inf.tu-dresden.de/

36]), а также систематизировал их в тексте диссертационной работы [8]. Этим текстом мы и воспользуемся в данном подразделе обзора родственных работ.

Как уже отмечалось, Укидом был реализован прототип NVM-ориентированной СУБД SOFORT. Это система с поколоночным хранением таблиц, квалифицируемая автором как СУБД категории HTAP (Hybrid Transactional and Analytical Processing). Как пишет Укид, SOFORT обеспечивает эффективное выполнение аналитических рабочих нагрузок, обеспечивая конкурентоспособную поддержку транзакционных приложений. По всей видимости, выбор такого неочевидного подхода (об этой неочевидности мы поговорим в начале разд. объясняется тесными связями Дрезденской группы вообще и ее руководителя Лехнера с проектом гибридной поколоночной un-memory СУБД HANA компании SAP. Кстати, влияние HANA прослеживается во всей работе Укида.

В целом, SOFORT – это СУБД с одноуровневой системой хранения, использующей как NVM, так и традиционную энергозависимую основную память. Применялся эмулятор NVM компании Intel. В своей работе Исмаил Укид последовательно описывает

- свою модель программирования использованием NVM;
- методы управления энергонезависимой памятью и ее распределения для нужд СУБД;
- реализованные в SOFORT методы управления транзакциями и восстановления баз данных после сбоев, а также
- разработанный фреймворк для тестирования NVM-ориентированного программного обеспечения.

В своем обзоре мы кратко обсудим только аспекты управления транзакциями.

Для управления транзакциями в SOFORT был адаптирован, оптимизирован и реализован многоверсионный оптимистический протокол (Multi-Version Optimistic Concurrency Control, MVOCC), впервые описанный в статье Пер-Оке Ларсона (Per-Ake Larson) и др.[10]

Основными отличиями подхода, используемого в SOFORT, от этого «традиционного» MVOCC являются (1) откат любой транзакции, пытающейся заблокировать строку таблицы, уже заблокированную другой транзакцией в несовместимом режиме и (2) особая, очень дешевая обработка фиксации транзакций, которые в своей рабочей фазе выполняли только операции чтения (для только читающих транзакций обеспечивается уровень snapshot isolation[11]).

В SOFORT каждая транзакция жестко ассоциируется с аппаратным потоком управления. Поэтому в системе поддерживается столько описателей транзакций, сколько потоков поддерживает используемая аппаратура. У каждого описателя транзакции имеются две части, одна из которых (persistent) располагается в энергонезависимой памяти, а другая (transient) – в обычной основной памяти. В первой части сохраняются признак фиксации транзакции, временная метка фиксации, а также наборы ссылок на строки таблиц, созданные и удаленные данной транзакцией. Информация из «постоянной» части описателя транзакции используется для выполнения индивидуальных откатов транзакции, а также заменяет журнал UNDO при восстановлении базы данных после сбоев.

Стоит заметить, что поколоночное хранение приводит к значительному усложнению выполнения всех транзакционных операций обновления таблиц и порождает дополнительные накладные расходы. Но нужно иметь в виду, что в компании SAP, под

[10] Per-Åke Larson, Spyros Blanas, Cristian Diaconu, Craig Freedman, Jignesh M. Patel, Mike Zwilling. High-performance concurrency control mechanisms for main-memory databases. Proceedings of the VLDB Endowment, vol. 5, issue 4, 2011, pp. 298-309.

[11] https://en.wikipedia.org/wiki/Snapshot_isolation

влиянием которой выполнялась работа Укида, ориентируются на транзакционную рабочую нагрузку с очень небольшим числом операций обновления баз данных (см. подраздел 3.1).

4. Факторы, ограничивающие «чистоту» архитектуры NVM-ориентированной СУБД

Стараниями, главным образом, молодежи мы получили более ясное представление о способах использования энергонезависимой памяти в архитектуре СУБД, о возникающих проблемах и возможных способах их решения. Однако все предпринятые попытки построения архитектуры NVM-ориентированных СУБД производились в условиях ряда разного рода ограничений, мешающих получить «чистую» архитектуру, на которую не влияли бы какие-либо внешние факторы.

По нашему мнению, на разработку архитектуры NVM-ориентированных СУБД влияют

- инфраструктурные ограничивающие факторы;
- недостаточная развитость аппаратных технологий NVM и/или недостаточная информированность сообщества баз данных со стороны вендоров;
- влияние решений, наиболее активно используемых в близкой, но принципиально отличающейся области in-memory СУБД.

4.1 Инфраструктурные ограничивающие факторы

Для разработки архитектуры любой программной системы, тем более такой сложной, как СУБД, тем более СУБД, основанной на новых принципах, требуется наличие инфраструктуры. Инфраструктура должна включать подготовленных и заинтересованных специалистов, с которыми можно было бы обсуждать общий облик будущей системы и/или альтернативные частные решения. Инфраструктура должна обеспечивать возможность быстрого построения прототипов системы, поддерживать необходимые процессы тестирования и отладки. Для этого необходимо наличие как готовых для использования аппаратно-программных инструментальных средств, так и программистов, способных быстро и достаточно качественно участвовать в разработке. Наконец, инфраструктура должна помогать находить источники финансирования новых проектов.

Как показывает обзор родственных работ в предыдущем разделе, почти все успешные проекты, связанные с использованием NVM в СУБД, опирались на наличие подобной инфраструктуры. Лучше всего ее обеспечивают исследовательские лаборатории компаний или университетов, постоянно выполняющие различные проекты по тематике, связанной с базами данных, обладающие необходимыми инструментальными средствами, содержащие высококвалифицированных специалистов и привлекающие к исследовательской работе студентов и аспирантов. Очевидными примерами являются группа баз данных в университете Карнеги-Меллон[12] и Дрезденская группа систем баз данных.

Но наличие инфраструктуры и ее особенности накладывают на разработку новой архитектуры свои ограничения. Главным образом, они связаны с тем, что трудно решиться на открытие совсем нового проекта по теме, которая не гарантирует абсолютного успеха. Это в полной мере соответствует актуальной, но не бесспорной тематике NVM-ориентированных СУБД. Понятно, что выполнение такого проекта в любом случае принесло бы пользу, но практическая значимость результатов не может быть известна заранее. Не говоря уже о том, что получить финансирование подобных

[12] https://db.cs.cmu.edu/

проектов совсем непросто, недостаточно убедительные результаты могут навредить имиджу лаборатории.

Поэтому естественно возникает желание «пристегнуть» новый проект к какому-то уже существующему, зарекомендовавшему себя и устойчиво финансируемому проекту. В нашем случае мы наблюдали привязку проекта Арулраджа к проекту СУБД Peloton в группе Карнеги-Меллон и привязку проекта Укида к проекту HANA в Дрезденской группе. Это приводит к нарушению чистоты архитектуры.

В группе Карнеги-Меллон основной целью проекта Peloton, в рамках которого работал Джой Арулрадж, является разработка самоуправляемой (self-driven) СУБД нового поколения, способной самостоятельно, без потребности во вмешательстве людей настраиваться на поддержку эффективного выполнения гибридных рабочих нагрузок. Под руководством явно увлеченного этой темой Энди Павло разработчики системы активно используют методы искусственного интеллекта и машинного обучения. При построении системы явно учитывается то, что она должна быть в состоянии настраиваться и на транзакционную, и на аналитическую рабочие нагрузки.

Нельзя не согласиться с важностью и актуальностью тематики самоуправляемых СУБД. Но, с другой стороны, по нашему мнению, Peloton отчасти связывает с тематикой NVM-ориентированных СУБД только то, что в ней базы данных хранятся в основной памяти. Самоуправляемость и поддержка гибридных рабочих нагрузок противоречат основной цели NVM-ориентированной СУБД – очень быстрая обработка транзакций и очень быстрое восстановление базы данных и работоспособности системы после сбоев. При совмещении обеих целей в одной разработке, фактически, происходит возврат к эпохе универсальных «безразмерных» систем, закат которой был убедительно показан Майклом Стоунбрейкером более 10 лет тому назад [37].

В работе Исмаила Укида предусловием, по-видимому, являлась ориентация на совместимость с производственной СУБД HANA компании SAP. С одной стороны, HANA является современной in-memory СУБД, основанных на современных, но уже проверенных технологиях. Но с другой стороны, эта система является (a) производственной, активно используемой внутри и вне SAP и (b) гибридной, рассчитанной на поддержку как аналитических, так и транзакционных рабочих нагрузок. Другими словами, это тоже универсальная in-memory СУБД. На мой взгляд, близость в HANA сильно повлияла на работу Укида, построенную им архитектуру NVM-ориентированной СУБД, на выбор алгоритмов и методов. Видимо, это оказалось полезным для профессиональной карьеры Укида как сотрудника SAP, но повредило «чистоте» его разработки.

Мы считаем, что нативную архитектуру одноуровневой СУБД можно разработать только с нуля, не подвергаясь влиянию особенностей инфраструктуры.

4.2 Ограничивающие предположения о характеристиках аппаратных средств NVM

Как показывает разд. 2 этой статьи, достоверную информацию о характеристиках не только ожидаемых аппаратных решениях NVM, но и уже производимых продуктах получить весьма затруднительно. Компании, разрабатывающие или уже производящие NVM в форм-факторе DIMM, не хотя преждевременно раскрывать их точные характеристики. Показатели разных видов NVM, публикуемые в открытых источниках, называют приблизительными сами авторы публикаций.

Данные, приводимые в табл. 1, относительно убедительно показывают только то, что (1) NVM на основе STT-RAM обладает показателями скорости доступа и стойкости, близкими к показателям DRAM и (2) NVM на основе PCM и ReRAM показывает более слабые характеристики. Однако судя по данным отдельных источников, скорость записи

STT-RAM в пять раз меньше скорости чтения ([3, 13]), а стойкость этого же вида NVM на один-два десятичных порядка уступает стойкости DRAM ([3, 5]).

Исходя из подобных недостаточно достоверных сведений, различные исследователи принимают разные пессимистические предположения о характеристиках ожидаемой энергонезависимой памяти.

- Некоторые из них полагают, что NVM является и навсегда останется существенно более медленной, чем DRAM (и, тем более, SSD) (возможно, так оно и есть для PCM), и поэтому следует использовать NVM в качестве еще одного слоя в иерархии систем хранения данных между DRAM и DDR.
- Другие исследователи полагают, что неустранимой особенностью всех видов байт-адресуемой NVM является существенная разница в скорости выполнения операций чтения и записи: запись медленнее чтения. Понятно, что это предположение сильно влияет на методы использования NVM в СУБД.
- Наконец, еще одним распространенным предположением является меньшая стойкость (endurance) NVM по сравнению как с DRAM, так и с DDR. Конечно, это предположение сдерживает энтузиазм по отношению к разработке одноуровневых NVM-ориентированных СУБД.

С одной стороны, поскольку в будущем, вполне возможно, на рынке станут доступными разные виды NVM в форм-факторе DIMM с разными характеристиками, и менее быстрая и стойкая энергонезависимая память будет стоить дешевле энергонезависимая память будет стоить дешевле быстрой и стойкой памяти (еще одно предположение!), могут оказаться востребованными различные архитектуры СУБД с использованием NVM.

- При наличии не слишком быстрой, но достаточно стойкой NVM может оказаться достаточно эффективная многоуровневая организация системы хранения DRAM-NVM. Это является естественной эволюцией технологии in-memory СУБД.
- При аналогичных показателях NVM вполне перспективен и путь к использованию иерархии DRAM-NVM-DDR (или SSD). Понятно, что этот подход является эволюцией традиционной технологией дисковых СУБД. В зависимости от показателей стойкости NVM эта память может использоваться в виде долговременно хранимого кэша между DRAM и DDR(SSD) или в виде части постоянного хранилища данных.
- Наконец, еще один путь к использованию NVM в форм-факторе DIMM состоит в том, чтобы вообще не использовать байт-адресуемую энергонезависимую память в архитектуре будущей СУБД. Как мы отмечали в разд. 2, в последнее время появились SSD (компании Intel на основе технологии Optane-3D XPoint и компании Samsung на основе технологии Z-NAND), которые обеспечивают очень высокую скорость блочных обменов. Поскольку в SSD компании Intel используется NVM типа PCM, в этих дисковых устройствах время произвольной записи блока почти не отличается от времени чтения. Конечно, при использовании таких SSD для хранения баз данных требуется пересмотр архитектуры СУБД (см., например, [11]), и в этой архитектуре может не оказаться места медленной байт-адресуемой СУБД.

Это все верно, но нас интересует чистая одноуровневая архитектура NVM-ориентированной СУБД, в которой применяются только байт-адресуемые DRAM и NVM. Чтобы обеспечить «чистоту» (nativeness) этой архитектуры, мы должны сделать оптимистические предположения относительно характеристик NVM:

- скорость произвольного доступа к NVM одинакова для операций чтения и записи и не уступает скорости доступа к DRAM;
- стойкость NVM не уступает стойкости DDR (SSD).

Возможно, эти предположения слишком оптимистичны (особенно второе), но они соответствуют наблюдаемым тенденциями развития технологии NVM.

4.3 Влияние методов, используемых в близких областях

NVM обладает чертами как устройств хранения данных во внешней памяти (энергонезависимость и следующая из этого долговременность хранения), так и традиционной основной памяти (прямая байтовая адресация). Это побуждает разработчиков архитектуры NVM-ориентированной СУБД пытаться использовать решения, успешно себя зарекомендовавшие в архитектурах дисковых и/или in-memory СУБД, даже если разработка новой архитектуры ведется с нуля. Однако NVM обладает важным отличием от устройств внешней памяти: это байт-адресуемая память прямого доступа. NVM принципиально отличается и от традиционной основной памяти: она энергонезависима и потому обеспечивает долговременное хранение данных.

Наверное, можно найти много примеров необоснованных заимствований решений, но в этой статье мы обсудим только два из них, относящихся к важным компонентам СУБД: индексации и управления транзакциями.

4.3.1 Индексация

Для организации индексов в дисковых СУБД наибольшей популярностью пользуются B+-деревья [38]. Узлами B+-деревьев являются блоки дисковой памяти, причем в промежуточных узлах хранятся линейные списки пар <значение ключа, номер дочернего блока>, таких что в соответствующем дочернем блоке (если он все еще не листовой) хранится список аналогичных пар со значением ключа, большим или равным значению ключа из родительской пары и меньшим значения ключа из следующей пары родительского списка. Листовые узлы B+-дерева хранят упорядоченные по возрастанию значений ключа пары <значение ключа, линейный список идентификаторов строк таблицы>.

В каждом промежуточном узле B+-дерева сохраняется, вообще говоря, много пар <значение ключа, номер дочернего блока>, поэтому дерево является сильно ветвистым и потому обладает небольшой глубиной. Из-за этого при поиске строк таблицы по заданному значению ключа, который всегда начинается с корневого узла, а заканчивается при достижении листового узла, содержащего требуемое значение ключа, приходится прочитать из дисковой в основную память небольшое число блоков.

Кроме того, для B+-деревьев поддерживается свойство полной сбалансированности, т.е. длина пути от корня до любого листа всегда одна и та же. Для этого используются операции расщепления (splitting) узла B+-дерева при переполнении узла и слияния (merging) соседних узлов (на одном уровне) при опустошении узла. Расщепление и слияние всегда начинаются от некоторого листового узла и двигаются вверх по дереву по направлению к корню, иногда, но редко приводя к глобальной перестройке дерева. Сбалансированность индекса на основе B+-дерева означает, что независимо от значения ключа, по которому производится поиск, стоимость поиска (оцениваемая как требуемое число обменов с внешней памятью) всегда одна и та же.

Листовые узлы B+-дерева связываются в двунаправленный список, что позволяет использовать соответствующие индексы для поиска строк таблицы с заданным диапазоном значений ключа по возрастанию или убыванию этих значений. Иногда для дополнительного ускорения поиска по диапазону значений ключа пары, хранимые в листовых узлах, упорядочивают по возрастанию или убыванию ключа.

Фундаментальные свойства B+-дерева – сильная ветвистость и полная сбалансированность – играют на пользу индексации, если индекс хранится во внешней памяти. При оценках стоимости операций над такими индексами учитывается только

требуемое число обменов с внешней памятью, а временные расходы на обработку содержимого узлов после их считывания в основную память не принимаются во внимание.

Во что превратится B+-дерево, если прямо перенести эту структуру в основную память? Легко видеть, что по своей сути это упорядоченный по возрастанию значений ключа список ссылок на строки индексируемой таблицы. Этот список хранится в листовых узлах. Верхняя часть поддерева служит только для того, чтобы можно было найти в этом списке ключ с заданным значением в этом листовом списке, не перебирая его последовательно. Фактически, это тоже иерархически организованный упорядоченный список упорядоченных подсписков. То, что каждый подсписок хранится внутри блока с последовательными адресами в линейной форме, никак не влияет на требуемое число обращений в памяти и операций сравнения.

Как справедливо отмечалось в [39], наличие поисковой подструктуры в B+-дереве, отделенной от списка ключей и ссылок на индексируемые строки таблицы, при хранении B+-деревьев в основной памяти приводит только к дополнительным накладным расходам памяти и процессорных ресурсов. Если уж применять B-деревья для организации индексов в основной памяти, то это должны быть классические B-деревья [40], в которых каждое существующее в индексируемой таблице значение ключа хранится только один раз (в каком-то промежуточном или листовом узле) и при нем сохраняется соответствующий набор ссылок на строки таблицы.

Можно сказать, что B-дерево в основной памяти – это один рекурсивно организованный упорядоченный список упорядоченных подсписков, или дерево, узлы которого являются списками. Если все подсписки содержат примерно одно и то же число значений ключа k, и глубина дерева равна n, то в худшем случае для поиска ключа с заданным значением придется потратить $O(k \times n)$ операций обращения к памяти и сравнения. Но блочная структура узлов дерева, сохраняющих подсписки, здесь не имеет никакого смысла. Каждый элемент любого подсписка может располагаться в отдельной области памяти, и все они могут связываться ссылками.

Тем не менее, как это не странно, индексы на основе именно B+-деревьев получили широкое распространение в in-memory СУБД. Изобретено большое число разновидностей B+-деревьев в основной памяти, и они активно заимствуются в новые архитектуры NVM-ориентированных СУБД именно в блочной форме. В частности, свои варианты B+-деревьев предлагают и Джой Арулрадж и Исмаил Укуд. По нашему мнению, подобные подходы к индексации таблиц в NVM-ориентированных СУБД не обоснованы, и методы индексации требуют дополнительного изучения (см. разд. 5).

4.3.2 Управление транзакциями

В последние десятилетия в сообществе разработчиков СУБД резко вырос интерес в многоверсионным методам управления транзакциями (MultiVersion Concurrence Control, MVCC). При применении протоколов MVCC каждое изменение объекта базы данных (как правило, строки таблицы) приводит к образованию новой версии этого объекта, а его предыдущая версия остается доступной по чтению в других транзакциях.

Основной причиной возросшей популярности MVCC является то, что при использовании классических двухфазных блокировочных протоколов (Two Phase Locking, 2PL) даже вполне безобидные транзакции, которые только читают данные, могут попасть в синхронизационный тупик, и из-за этого система может их насильственно откатить. Примером может служить следующая последовательность операций над объектами базы данных o_1 и o_2, выполняемых в транзакциях T_1 и T_2: $R_{T_1}(o_1), W_{T_2}(o_2), R_{T_1}(o_2), W_{T_2}(o_1)$. Если используется какой-либо протокол категории MVCC, то первая транзакция прочитает без всякой задержки на синхронизацию то состояние объекта o_2, в котором он

находился до изменения транзакцией T_2, т.е. план выполнения смеси транзакций T_1 и T_2 изменится на допустимый сериализацией план $R_{T_1}(o_1), R_{T_1}(o_2), W_{T_2}(o_2), W_{T_2}(o_1)$.

Мы отложим более подробное обсуждение протоколов MVCC и связанных с ними преимуществ и проблем до разд. 5, а здесь поговорим об их применении в in-memory и NVM-ориентированных СУБД. Похоже, что во всех современных in-memory СУБД используется какой-либо вариант протокола MVCC. Наиболее известными являются базовые подходы многоверсионного протокола с упорядочиванием по временным меткам (MultiVersion Timestamp Ordering, MVTO), двухверсионный двухфазный блокировочный протокол (Two-Version 2PL, 2V2PL) и оптимистический многоверсионный протокол (MultiVersion Optimistic Concurrency Control, MVOCC).

Краткий обзор этих протоколов можно найти, например, в [41][13]. В этой же статье говорится о том, что все эти протоколы были реализованы в СУБД Peloton и был произведен их экспериментальный анализ с использованием бенчмарков YCSB и TPC-C. На основе содержимого статьи можно сделать следующие наблюдения:

- разработчики Peloton так и не выбрали базовый протокол управления транзакциями для своей системы;
- у всех испытанных протоколов имеются свои достоинства и недостатки, проявляемые при различных параметрах рабочей нагрузки;
- видимо, поэтому в работах [4-6] расплывчато говорится об использовании в архитектуре NVM-ориентированной СУБД протокола MVCC без уточнения его разновидности.

По нашему мнению, это означает, что Арулрадж и его научный руководитель Павло в действительности не представили полную архитектуру NVM-ориентированной СУБД, поскольку специфические черты конкретного механизма управления транзакциями накладывают отпечаток на многие другие компоненты СУБД.

И это видно на примере работы Исмаила Укуда [8]. В своей архитектуре он выбрал протокол MVOCC. Выбор именно этого протокола серьезно не оправдывается. Протокол сложный, реализация технически трудная. Возможно, для диссертации это только плюс, а для «чистой» архитектуры NVM-ориентированной СУБД – минус, по нашему мнению. Сошлемся на цитату из книги Джема Грея, полностью приведенную в конце подраздела 3.1: «…простые системы работают, а сложные нет».

5. Набросок архитектуры «чистой» NVM-ориентированной СУБД

В настоящее время у автора отсутствует полная архитектура «чистой» NVM-ориентированной СУБД. Даже эскизы архитектуры недостаточно технически проработана. Полностью отсутствует какая-либо реализация. Однако в условиях недостаточного внимания исследователей к общим архитектурным вопросам организации NVM с использованием одноуровневой энергонезависимой памяти приводимые в этом разделе наблюдения и рассуждения могут оказаться полезными при будущей полномасштабной разработке.

Начнем раздел со сводки базовых предположений, которыми руководствуется автор.

- NVM обладает всеми «оптимистическими» характеристиками: скорость чтения и записи одна и та же и не меньше, чем у DRAM; стойкость не меньше, чем у DDR (или хотя бы SSD).

[13] Мы не включали эту статью в число ранее указанных публикаций Арулраджа, потому что в ней ничего не говорится о NVM-ориентированных СУБД.

- Целевой компьютер содержит требуемое число ядер или потоков управления, поддерживаемых аппаратурой, а также достаточные объемы энергозависимой и энергонезависимой основной памяти.
- Внешняя память совсем не используется.
- Вся серверная часть системы выполняется на одном компьютере, в энергонезависимой основной памяти которого сохраняется баз данных. Сеть используется только для взаимодействия с сервером клиентских частей приложений.
- Архитектура рассчитана на поддержку только транзакционных рабочих нагрузок.
- При использовании аппаратных средств не допускается никакая виртуализация. СУБД сама следит за распределением энергонезависимой основной памяти. Каждой транзакции жестко соответствует отдельное ядро процессора или поток управления, поддерживаемый аппаратурой.

С учетом этих предположений рассмотрим предлагаемые архитектурные решения.

5.1 Распределение энергонезависимой памяти

Для упрощения решения, в том числе, преодоления проблемы внешней фрагментации памяти за счет допущения ограниченной внутренней фрагментации предлагается основывать распределение памяти на классическом методе двоичных близнецов (buddy system) [42]. При применении этого метода при запросе k байт памяти всегда выделяется участок размером в 2^nбайт, где 2^n – ближайшая к k сверху степень двойки.

По нашему мнению, для распределения энергонезависимой памяти в контексте NVM-ориентированной СУБД больше подходит именно метод двоичных близнецов, а не, например, фибоначчиевых близнецов (Fibonacci system)[14], когда при запросе k байт памяти выделяется участок, размер которого равен ближайшему сверху числу Фибоначчи. Это связано с тем, что фактическим стандартом современной индустрии микропроцессоров общего назначение является использование в кэш-памяти блоков (cache line) размером 64 байта. Поэтому мы предполагаем, что размер динамически запрашиваемой энергонезависимой в NVM-ориентированной СУБД никогда не будет меньше 2^6, а в основном будут запрашиваться участки памяти именно этого размера.

Понятно, что память может запрашиваться в любой выполняемой транзакции, которой соответствует ядро процессора или поток управления, поддерживаемый аппаратурой. Чтобы избежать потребности в синхронизации параллельно выполняемых потоков управления, свободная энергонезависимая память заранее делится поровну между всеми рабочими потоками управления (worker), и в каждом потоке работает собственный компонент распределения памяти в своей части общей кучи.

Представляется, что в основном свободные участки памяти будут запрашиваться явно путем обращения к «системным» вызовам типа malloc. Но потребуются и неявные запросы памяти («по требованию») при расширении сегментов, требуемых для поддержки индексов (см. 5.3). Нужно, чтобы в случае, когда прерывание по отсутствию в NVM страницы виртуальной памяти соответствующего сегмента возникает при выполнении транзакции в некотором ядре процессора, операционная система брала соответствующий фрейм NVM из «подкучи» этого же ядра (аппаратного потока управления).

5.2 Хранение таблиц и управление транзакциями

Для упрощения архитектуры и по другим соображениям, обсуждаемым в п. 5.2.2, мы предлагаем использовать в «чистой» архитектуре NVM-ориентированной СУБД разновидность протокола 2V2PL. Напомним, что 2V2PL предполагает хранение двух

[14] Daniel S. Hirschberg. A class of dynamic memory allocation algorithms. Communications of the ACM, vol. 16, issue 10, 1973, pp. 615-618.

копий каждой строки каждой таблицы базы данных: текущей версии, созданной последней зафиксированной транзакцией, и измененной версии, созданной еще не зафиксированной транзакцией. Естественно, это влияет на организацию хранения таблиц в NVM.

5.2.1 Хранение таблиц

Каждая таблица сохраняется в энергонезависимой основной памяти в виде списка строк, т.е. таблицы хранятся, вообще говоря, по строкам (с оговоркой, приводимой ниже). Выборку строк их таблицы можно выполнять либо путем последовательного просмотра этого списка, либо через индекс (см. под

Транзакционные базы данных по определению являются изменчивыми, поэтому память, запрошенная при вставке строки или при создании первой копии, не освобождается до выполнения операции удаления этой строки. В первом приближении схема выглядит следующим образом. При вставке строки всегда запрашивается участок памяти размером не менее 64 байта. Этот участок представляет новую строку в списке строк таблицы. Точный размер запрашиваемого участка памяти определяется тем, что в нем должны поместиться две копии строки таблицы и служебная информация (ссылки по списку строк и информация, требуемая для управления транзакциями (см. п. 5.2.2)).

Если таблица содержит столбцы большого размера (например, длинные varchar) для их хранения используются отдельные участки памяти требуемого размера, для доступа к которым в основном месте хранения строк, вместо самих данных записываются указатели на области их расположения. Возможно, потребуется оптимизация при таком отдельном хранении столбцов, свойственная поколоночным СУБД (устранение дубликатов и/или сжатие), но для целей данной статьи это несущественно.

5.2.2 Управление транзакциями

На решение использовать для управления транзакциями в «чистой» архитектуре протокол 2V2PL во многом повлияли статьи [41, 43], в которых на основе убедительных экспериментов было показано, что ни один из известных протоколов управления транзакциями даже на простых транзакционных бенчмарках не демонстрирует преимуществ. Видимо, это так и есть, и с использованием имеющихся подходов невозможно обеспечить горизонтальное масштабирование как in-memory, так и NVM-ориентированных СУБД при значительном возрастании числа ядер процессора. С другой стороны, основной задачей разработки «чистой» архитектуры NVM-ориентированной СУБД является обеспечение значительного повышения скорости обработки транзакций при сохранении максимальной простоты решений, а 2V2PL, по нашему мнению, гораздо проще, чем MVTO и, тем более, MVOCC.

Замечание. После публикаций очень эффектных статей о проекте H-Store[15] и столь же эффектном появлении коммерческой СУБД VoltDB[16] у автора данной статьи создалось впечатление, что удалось найти путь к горизонтальному масштабированию транзакционных СУБД путем использования массивно-параллельных компьютерных архитектур и разделения данных между узлами в духе shared-nothing [44]. Но даже и в этой чрезмерно оптимистической статье высказывались сомнения о возможности эффективной и масштабируемой поддержке распределенных транзакций.

[15] Robert Kallman, Hideaki Kimura, Jonathan Natkins, Andrew Pavlo, Alexander Rasin, Stanley B. Zdonik, Evan P. C. Jones, Samuel Madden, Michael Stonebraker, Yang Zhang, John Hugg, Daniel J. Abadi. H-store: a high-performance, distributed main memory transaction processing system. Proceedings of the VLDB Endowment, vol. 1, issue 2, 2008, pp. 1496-1499.

[16] https://en.wikipedia.org/wiki/VoltDB

Эти сомнения подтвердила очень полезная статья [45], в которой на основе экспериментов показано, что в настоящее время отсутствует метод построения СУБД, обеспечивающий горизонтальное масштабирование системы при наличии транзакционных рабочих нагрузок, содержащих распределенные транзакции. Одновременно эта статья укрепила наше мнение, высказанное в [11], о том, что «чистая» архитектура NVM-ориентированной СУБД должна основываться на одномашинной многоядерной аппаратной архитектуре. Попытки добиться горизонтальной масштабируемости оставим на будущее.

В общих чертах предлагаемый вариант 2V2PL выглядит следующем образом. Поскольку в каждом ядре (аппаратном потоке) в каждый момент времени выполняется только одна транзакция, можно содержать в энергонезависимой памяти по одному описателю транзакции для каждого ядра (потока). У каждой транзакции имеется уникальный идентификатор, не используемый повторно для будущих транзакций. Идентификатор транзакции сохраняется в описателе транзакции. Когда транзакция создает новую версию некоторой строки некоторой таблицы (вставляет новую или изменяет существующую строку), она записывает в служебную часть хранилища этой копии свой идентификатор транзакции и ссылку на свой описатель, а также обнуляет поле идентификатора транзакции в предыдущей копии строки, если она существует. Последним действием при фиксации транзакции, после которого ее уже нельзя откатить, является обнуление идентификатора в описателе транзакции. Другими словами, признаком наличия измененной (незафиксированной) копии строки является совпадение сохраненного при ней идентификатора транзакции с идентификатором, находящимся в описателе транзакции. В общей служебной части хранилища строки содержится неотрицательный счетчик, значение соответствует числу незафиксированных транзакций, прочитавших эту копию, и признак выполнения операции записи.

Тогда в рабочей фазе любой транзакции при выполнении операции чтения строки таблицы увеличивается на единицу значение счетчика последней зафиксированной копии (эта операция атомарная) и считывается строка. В описателе транзакции сохраняется эта строка, а также порядковый номер ее версии. В конце операции чтения счетчик уменьшается на единицу. Если в той же транзакции повторно выполняется чтение той же строки, проверяется, что у нее не изменился номер версии. Если он изменился, то чтение все равно производится, но логика приложения об этом оповещается и принимает решение о продолжении или откате транзакции.

При выполнении операции записи проверяется признак выполнения операции записи той же строки другой транзакцией. Если признак установлен, транзакция откатывается, иначе признак устанавливается (проверка и установка признака – атомарная операция). Далее проверяется значение счетчика последней зафиксированной копии. Если он равен нулю, проверяется, не существует ли копия, образованная еще не зафиксированной транзакцией. Если такая копия не существует, запись производится на место предпоследней зафиксированной копии, а в служебную область этой копии пишутся ссылка на описатель и идентификатор транзакции, в которой выполняется операция записи (образуется незафиксированная, измененная копия. Иначе снимается признак выполнения операции записи и транзакция откатывается. При благополучном исходе в описателе транзакции запоминается предыдущее состояние строки, номер образованной версии и адрес для обратной записи при восстановлении. В конце операции признак выполнения операции записи снимается.

Фиксация транзакции происходит мгновенно: обнуляется идентификатор транзакции в ее описателе. После этого в рабочем потоке можно выполнять новую транзакцию.

Индивидуальные откаты транзакций выполняются путем обратной записи изменений. Поскольку это касается только операций записи, обратную запись при восстановлении можно выполнять напрямую по сохраненным в описателях транзакции указателям.

Восстановление после сбоя, приведшего к перезапуску системы можно производить, пользуясь информацией, сохраненной в описателях транзакций, в которых к моменту сбоя содержатся ненулевые идентификаторы. Хронологически порядок обратных записей для какой-либо строки поддерживать не обязательно: если номер версии очередной копии строки для восстановления оказывается меньше номера копии, содержащегося в области хранения строки, обратную запись выполнять не нужно.

5.3 Организация индексов

Нам неизвестны точные характеристики запросов к базам данных, встречающиеся в типичных современных транзакционных рабочих нагрузках. Но практически во всех известных случаях для измерения и/или сравнения производительности транзакционных СУБД используется бенчмарк TPC-C. Мы принимаем это как свидетельство о реалистичности этого бенчмарка.

Почти все запросы к базе данных в транзакциях TPC-C опираются на условия вида $a = b$, где a – имя столбца одной таблицы из раздела FROM запроса, а b – это либо константа, либо имя столбца другой таблицы из раздела FROM (для запросов с соединениями). При этом чаще всего в условиях используются имена столбцов первичных ключей таблиц.

Поэтому прежде всего в транзакционной NVM-ориентированной СУБД требуются индексы, обеспечивающие быстрый прямой доступ к строкам таблиц по значению указанного столбца. Очевидно, что для этого лучше всего приспособлены методы, основанные на хешировании значений ключа поиска, и по нашему мнению наиболее просто и эффективно можно реализовать индекс на основе варианта метода линейного хеширования [46]. Основная часть таблицы должна храниться в расширяемом сегменте виртуальной памяти, страницы которого отображаются на NVM.[17]

Представляется, что для каждого значения свертки будет поддерживаться основной бакет размером 64 байта. Если до потребности в расщеплении элемента таблицы список элементов <ключ, ссылка на строку таблицы> перестанет помещаться в уже выделенных бакетах, то будет запрашиваться еще один бакет, и он будет привязываться к списку уже существующих бакетов (так может быть при индексировании таблицы по столбцам с длинными значениями, которые, тем не менее, невыгодно хранить отдельно). Критерием расщепления является достижение установленного предела длины списка с одинаковым значением свертки.

Поскольку трудно отказаться от поисковых операций уровня SQL с указанием условий поиска вида $c_1 < a < c_2$, где a – имя столбца таблицы из раздела FROM запроса, а c_1 и c_2 – константы (запрос по диапазону, range query), придется обеспечивать другой вид индексов, скорее всего, на основе B-деревьев. На наш взгляд, стоит рассмотреть два варианта. В первом варианте все узлы дерева представляют списки бакетов по 64 байта, а критерием расщепления или слияния соседних узлов является длина списка. Во втором варианте каждое B-дерево располагается в отдельном расширяемом сегменте виртуальной памяти, а узел дерева – линейный список элементов, располагающийся в пределах одной страницы сегмента. Для выбора варианта требуются эксперименты, измерения и сравнения.

Дополнительного изучения требуют вопросы организации параллельного изменения индексов, а также обратного изменения индексов при откате транзакций.

[17] По мнению автора, для NVM-ориентированной СУБД более подходящей была бы виртуальная память с небольшими страницами (например, 512 байт). В настоящее время ОС Linux допускает минимальный размер страницы в 4K байт.

5.4 Оптимизация запросов

В этом подразделе мы ограничимся лишь несколькими замечаниями. Во-первых, аспекты оптимизации SQL-запросов в NVM-ориентированных СУБД к настоящему времени исследованы явно недостаточно. Публикации, посвященные этой теме, практически отсутствуют, хотя, например, в заключении статьи [47] уже отмечалась потребность в таких исследованиях.

Насколько можно судить по текущим публикациям (например, обзорной статье [48]), в существующих in-memory СУБД используются методы оптимизации запросов, не отличающиеся от применяемых в традиционных дисковых СУБД. Обоснований этого консерватизма найти не удается. Хотелось бы при разработке «чистой» архитектуры NVM-ориентированной СУБД хорошо разобраться с требуемыми методами оптимизации запросов.

Во-вторых, мы ориентируемся на транзакционный режим использования СУБД. Очевидно, что в NVM-ориентированной СУБД транзакции должны обрабатываться очень быстро, поскольку в принципе отсутствуют обмены с устройствами внешней памяти. Чтобы обеспечить потенциально возможную пропускную способность NVM-ориентированной транзакционной СУБД, нужно избегать любых задержек при выполнении транзакций, в том числе, и коммуникаций с клиентскими рабочими станциями. Поэтому традиционным способом организации транзакционных приложений баз данных является перенос всей логики приложения на сторону сервера баз данных обычно в виде хранимых процедур.

Здесь мы не будем подробно останавливаться на методах программирования хранимых процедур в «чистой» NVM-ориентированной СУБД. Заметим лишь, что код хранимой процедуры, в которой содержится логика транзакции, должен выполняться в той же ядре (или аппаратном потоке управления), в котором выполняются все остальные системные программы, нужные для поддержки выполнения транзакции. Для всей СУБД поддерживается одно адресное пространства. Поэтому для программирования хранимых процедур должны использоваться безопасные, компилируемые в машинные коды (для поддержки эффективности), языки программирования (например, язык Rust[18]).

В этих хранимых процедурах, безусловно, должно допускаться использование языка SQL. Но компиляция и оптимизация декларативных оператором SQL должна производиться тогда же, когда выполняется компиляция кода хранимой процедуры. Это должно делаться раньше того времени, когда система начнет выполнять реальную транзакционную рабочую нагрузку. Поэтому должна производиться полноценная оптимизация с учетом специфики среды NVM.

Конечно, при оптимизации нужно учитывать стоимость генерируемых планов запросов, и для этого придется использовать статистику базы данных на момент оптимизации. Эта статистика может измениться при выполнении транзакционной рабочей нагрузки, и выбранный план может стать далеким от оптимального. NVM-ориентированная СУБД, видимо, сможет обнаружить этот факт в рабочем режиме, но выполнить повторную оптимизацию, не нарушая установившийся режим обработки транзакций, невозможно. Известные приемы, в том числе, адаптивная (самообучаемая) оптимизация запросов[19] здесь неприменимы. Требуется дополнительное исследование.

В-третьих, важно понять, какой SQL нужно поддерживать в NVM-ориентированных СУБД. С одной стороны, очевидно, что полный набор возможностей SQL,

[18] https://www.rust-lang.org/

[19] Идеи адаптивной оптимизации запросов можно наблюдать уже в ранних публикациях о System R, но первой хорошо проработанной работой мы считаем статью Volker Markl, Guy M. Lohman, Vijayshankar Raman. LEO: An autonomic query optimizer for DB2. IBM Systems Journal, vol. 42 , issue 1, 2003, pp. 98 – 106.

специфицированных в стандарте, здесь не потребуется. С другой стороны, волевым усилием ограничить некоторое подмножество стандарта явно невозможно. Требуется тщательная и кропотливая работа для выбора подмножества стандарта, которое бы действительно требовалось в специализированной транзакционной СУБД.

6. Заключение

Энергонезависимая байт-адресуемая основная память становится (или уже стала) реальностью. Характеристики разных видов NVM, доступные в открытом доступе, позволяют рассчитывать, что доступная в ближайшем будущем энергонезависимая основная память будет обладать скоростью, не меньше, чем DRAM, и стойкостью, сопоставимой со стойкостью HDD (или хотя бы SSD). Поэтому исследования архитектур СУБД с одноуровневой системой хранения становятся более чем актуальными.

Однако, как показывает анализ литературных источников, в сообществе баз данных этой тематике все еще уделяется недостаточное внимание. Даже наиболее удачные опубликованные исследования выполнялись при наличии ряда ограничений, не позволяющих получить «чистую» архитектуру транзакционной NVM-ориентированной СУБД, при разработке которой принималось бы во внимание только стремление к достижению максимально высоких показателей скорости выполнения транзакций и пропускной способности системы.

В статье приводится набросок такой архитектуры, в котором выделяются важные аспекты распределения памяти, организации хранения таблиц и индексов, управления транзакциями и оптимизации запросов. Почти во всех предложениях автора отмечается потребность в дополнительных исследованиях.

Список литературы / References

[1] Chris Mellor. Escaping the DRAM price trap: Storage Class Memory, what it is and why it matters. Blocks & Files, 2019. Available at: https://blocksandfiles.com/2018/11/28/2019-the-year-of-storage-class-memory/, accessed 02-06-2019

[2] Anton Shilov. Samsung Ships First Commercial Embedded MRAM (eMRAM) Product. AnandTech, March 6, 2019. Available at: https://www.anandtech.com/show/14056/samsung-ships-first-commercial-emram-product, accessed 02-06-2019.

[3] Lucian Armasu. Intel STT-MRAM Technology Is Ready for Mass Production. Tom's Hardware, February 21, 2019. Available at: https://www.tomshardware.com/news/intel-stt-mram-mass-production,38665.html, accessed 02-06-2019.

[4] Joy Arulraj. The Design and Implementation of a Non-Volatile Memory Database Management System. PhD Thesis, Computer Science Department, School of Computer Science, Carnegie Mellon University, 2018, 51 p.

[5] Joy Arulraj, Andrew Pavlo. How to Build a Non-Volatile Memory Database Management System. In Proc. of the 2017 ACM International Conference on Management of Data, 2017, pp. 1753-1758.

[6] Joy Arulraj, Andrew Pavlo. Non-Volatile Memory Database Management Systems. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2019, 173 p.

[7] Joy Arulraj. Data Management on Non-Volatile Memory. Award Talk. In Proc. of the 2019 International Conference on Management of Data, 2019, p. 1114.

[8] Ismail Oukid. Architectural principles for database systems on storage-class memory. PhD.thesis, Technische Universität Dresden, 2017, 189 p.

[9] Oukid, D. Booss, A. Lespinasse, W. Lehner, T. Willhalm, and G. Gomes. Memory management techniques for large-scale persistent-main-memory systems. Proceedings of the VLDB Endowment, Vol. 10, Issue 11, 2017, pp. 1166-1177.

[10] Oukid and W. Lehner. Data structure engineering for byte-addressable non-volatile memory. In Proc. of the 2017 ACM International Conference on Management of Data, 2017, pp. 1759-1764.

[11] С.Д. Кузнецов. Новые устройства хранения данных и их влияние на технологию баз данных. Программная инженерия, no. 4. 2018, стр. 147–155 / Sergey D. Kuznetsov. New Storage Devices

and the Future of Database Management. Baltic Journal of Modern Computing, Vol. 6, No. 1. 2018, pp. 1-12. DOI: 10.22364/bjmc.2018.6.1.01.

[12] Байтоадресуемая энергонезависимая память и СУБД. Презентация Андрея Николаенко на Tarantool Conference 2018 / Byte-addressable non-volatile memory and DBMS. Presentation by Andrey Nikolayenko at the Tarantool Conference 2018. Available at: https://ibs.ru/media/media/baytoadresuemaya-energonezavisimaya-pamyat-i-subd/, accessed 09.06.2019 (in Russian).

[13] Sparsh Mittal, and Jeffrey S. Vetter A Survey of Software Techniques for Using Non-Volatile Memories for Storage and Main Memory Systems. IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 5, 2016, pp. 1537-1550.

[14] Micron Technology. Available at: https://www.micron.com/, accessed 18-06-2019.

[15] Panasonic and United Microelectronics Corporation Agreed to Develop Mass Production Process for Next Generation ReRAM, Feb 01, 2017. Available at: https://news.panasonic.com/global/press/data/2017/02/en170201-3/en170201-3.html, accessed 18-06-2019.

[16] Crossbar. Available at: https://www.crossbar-inc.com/company/about-crossbar/, accessed 18-06-2019.

[17] Everspin Technologies Available at: https://www.everspin.com/, accessed 18-06-2019.

[18] Crocus Technology. Available at: https://crocus-technology.com, accessed 22-06-2019.

[19] Crocus Nano Electronics. Available at: http://crocusnano.com, acccessed 22-06-2019.

[20] L. Wei, J. G. Alzate, U. Arslan, J. Brockman, N. Das, K. Fischer, T. Ghani,O. Golonzka, P. Hentges, R. Jahan, P. Jain, B. Lin, M. Meterelliyoz, J. O'Donnell,C. Puls, P. Quintero, T. Sahu, M. Sekhar, A. Vangapaty, C. Wiegand, F. Hamzaoglu. 7Mb STT-MRAM in 22FFL FinFET Technology with 4ns Read Sensing Time at 0.9V Using Write-Verify-Write Scheme and Offset-Cancellation Sensing Technique. In Proc. of the IEEE International Solid-State Circuits Conference (ISSCC), 2019, pp. 214-216.

[21] P. Jain, U. Arslan, M. Sekhar, B. C. Lin, L. Wei, T. Sahu, J. Alzate-vinasco,A. Vangapaty, M. Meterelliyoz, N. Strutt, A. B. Chen, P. Hentges, P. A. Quintero,C. Connor, O. Golonzka, K. Fischer, F. Hamzaogl. 3.6Mb 10.1Mb/mm2Embedded Non-Volatile ReRAM Macro in 22nm FinFET Technology with Adaptive Forming/Set/Reset Schemes Yielding Down to 0.5V with Sensing Time of 5ns at 0.7. In Proc. of the IEEE International Solid-State Circuits Conference (ISSCC), 2019, pp. 212-214.

[22] Stratis D. Viglas. Data Management in Non-Volatile Memory. In Proc. of the 2015 ACM SIGMOD International Conference on Management of Data, 2015, pp. 1707-1711

[23] Naveed Ul Mustafa, Adrià Armejach, Ozcan Ozturk, Adrián Cristal, Osman S. Unsal. Implications of non-volatile memory as primary storage for database management systems. In Proc. of the 2016 International Conference on Embedded Computer Systems: Architectures, Modeling and Simulation (SAMOS), 2016, pp. 164-171

[24] Mihnea Andrei, Christian Lemke, Günter Radestock, Robert Schulze, Carsten Thiel, Rolando Blanco, Akanksha Meghlan, Muhammad Sharique, Sebastian Seifert, Surendra Vishnoi, Daniel Booss, Thomas Peh, Ivan Schreter, Werner Thesing, Mehul Wagle, Thomas Willhalm. SAP HANA Adoption of Non-Volatile Memory. Proceedings of the VLDB Endowment, vol. 10, no. 12, 2017, pp. 1754-1765.

[25] Alexander van Renen, Viktor Leis, Alfons Kemper, Thomas Neumann, Takushi Hashida, Kazuichi Oe, Yoshiyasu Doi, Lilian Harada, Mitsuru Sato. Managing non-volatile memory in database systems. In Proc. of the 2018 ACM SIGMOD International Conference on Management of Data, 2018, pp. 1541-1555.

[26] Joy Arulraj, Andrew Pavlo, Subramanya R. Dulloor. Let's Talk About Storage & Recovery Methods for Non-Volatile Memory Database Systems. In Proc. of the 2015 ACM SIGMOD International Conference on Management of Data, pp. 707-722

[27] Joy Arulraj, Matthew Perron, Andrew Pavlo. Write-Behind Logging. Proceedings of the VLDB Endowment, vol. 10, no. 4, pp. 337-348, 2017.

[28] J. Arulraj, J. Levandoski, U. F. Minhas, and P.-A. Larson. BzTree: A high-performance latch-free range index for non-volatile memory. Proceedings of the VLDB Endowment, vol. 11, no. 5, pp. 553-565, 2018.

[29] Joy Arulraj, Andrew Pavlo, Krishna Teja Malladi. Multi-Tier Buffer Management and Storage System Design for Non-Volatile Memory. arXiv:1901.10938, 2019.

[30] J. Gray, P. McJones, M. Blasgen, B. Lindsay, R. Lorie, T. Price, F. Putzolu, and I. Traiger. The recovery manager of the system R database manager. ACM Computing Surveys, vol. 13, no. 2, 1981, pp. 223-242.

[31] Philip A. Bernstein, Vassos Hadzilacos, Nathan Goodman. Concurrency control and recovery in database systems. Addison-Wesley, 1987, 370 p.

[32] Ismail Oukid, Daniel Booss, Wolfgang Lehner, Peter Bumbulis, Thomas Willhalm. SOFORT: A Hybrid SCM-DRAM Storage Engine for Fast Data Recovery. In Proc. of the Tenth International Workshop on Data Management on New Hardware (DaMoN'14), 2014.

[33] Ismail Oukid, Wolfgang Lehner, Thomas Kissinger, Thomas Willhalm, Peter Bumbulis. Instant Recovery for Main-Memory Databases. In Proc. of the 7th Biennial Conference on Innovative Data Systems Research (CIDR'15), 2015.

[34] Ismail Oukid, Johan Lasperas, Anisoara Nica, Thomas Willhalm, Wolfgang Lehner. FPTree: A Hybrid SCM-DRAM Persistent and Concurrent B-Tree for Storage Class Memory. In Proc. of the 2016 ACM SIGMOD International Conference on Management of Data, 2016, pp. 371-386.

[35] Ismail Oukid, Wolfgang Lehner. Towards a Single-Level Database Architecture on NVM. Presentation abstract. In Proc. of the 8th Annual Non-Volatile Memories Workshop, 2017.

[36] Ismail Oukid. Architectural Principles for Database Systems on Storage-Class Memory. This paper is a summary of the author's PhD thesis of the same title. Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn, 2019.

[37] Michael Stonebraker, Ugur Cetintemel. «One Size Fits All»: An Idea Whose Time Has Come and Gone. In Proc. of the 21st International Conference on Data Engineering, 2005, pp. 2-11.

[38] Douglas Comer. Ubiquitous B-Tree. ACM Computing Surveys, vol, 11, issue 2, 1979, pp. 121-137.

[39] Tobin J. Lehman, Michael J. Carey. A Study of Index Structures for Main Memory Database Management Systems. In Proc. of the 1986 ACM SIGMOD international conference on Management of data, 1986, pp. 239-250.

[40] Rudolf Bayer. R. Bayer, E. McCreight. Organization and Maintenance of Large Ordered Indexes. Acta Informatica, vol. 1, issue 3, 1972, pp. 173-189.

[41] Yingjun Wu, Joy Arulraj, Jiexi Lin, Ran Xian, Andrew Pavlo. An empirical evaluation of in-memory multi-version concurrency control. Proceedings of the VLDB Endowment, vol. 10, issue 7, 2017, pp. 781-792.

[42] Kenneth C. Knowlton. A fast storage allocator. Communications of the ACM, vol. 8, issue 10, 1965, pp. 623-624.

[43] Xiangyao Yu, George Bezerra, Srinivas Devadas, Michael Stonebraker, Andrew Pavlo. Staring into the Abyss: An Evaluation of Concurrency Control with One Thousand Cores. Proceedings of the VLDB Endowment, vol. 8, issue 3, 2014, pp. 209-220.

[44] Кузнецов С.Д. Транзакционные параллельные СУБД: новая волна. Труды ИСП РАН, том 20, 2011, стр. 189-251 / Sergey D. Kuznetsov. Transactional Massive-Parallel DBMSs: A New Wave. Trudy ISP RAS/Proc. ISP RAS, vol. 20, 2011, pp. 189-251 (in Russian).

[45] Rachael Harding, Dana Van Aken, Andrew Pavlo, Michael Stonebraker. An Evaluation of Distributed Concurrency Control. Proceedings of the VLDB Endowment, vol. 10, issue 5, 2017, pp. 553-564.

[46] Witold Litwin. Linear Hashing: A New Tool for File and Table Addressing. In Proc. of the 6th International Conference on Very Large Data Bases, 1980, pp. 212-223.

[47] S. Pelley, T. F. Wenisch, and K. LeFevre. Do query optimizers need to be SSD-aware? In Proc. of the 2nd International Workshop on Accelerating Data Management Systems using Modern Processor and Storage Architecture, 2011.

[48] F. Faerber, A. Kemper, P. Å Larson, J. Levandoski, T. Neumann, and A. Pavlo. Main Memory Database Systems. Foundations and Trends in Databases, vol. 8, No. 1-2, 2016, pp. 1–130.

Информация об авторе / Information about author

Сергей Дмитриевич КУЗНЕЦОВ – доктор технических наук, профессор, главный научный сотрудник ИСП РАН, профессор кафедр системного программирования МГУ, МФТИ и ВШЭ, старший научный сотрудник РЭУ им. Г.В. Плеханова. Научные интересы: управление данными, архитектуры систем управления данными, модели и языки данных, управление транзакциями, оптимизация запросов.

Sergey Dmitrievich KUZNETSOV - Doctor of Technical Sciences, Professor, Chief Researcher at ISP RAS, Professor at the Departments of System Programming of MSU, MIPT, and HSE, Senior Researcher at REU. Research interests: data management, architectures of data management systems, data models and languages, transaction management, query optimization.