ПРОГРАММИРОВАНИЕ, 2023, № 1, с. 3–31

АНАЛИЗ ДАННЫХ

УДК 004.6

АНАЛИТИКА В РЕАЛЬНОМ ВРЕМЕНИ: ПРЕИМУЩЕСТВА, ОГРАНИЧЕНИЯ И КОМПРОМИССЫ

© 2023 г. С. Д. Кузнецов[a,b,c,d,*] (ORCID: 0000-0002-8257-028X), П. Е. Велихов[e,] (ORCID: 0000-0002-0644-8047), Ц. Фу[f,***] (ORCID: 0000-0003-0244-1718)**

[a]*Институт системного программирования им. В.П. Иванникова РАН, 109004 Москва, ул. А. Солженицына, д. 25, Россия*

[b]*Московский государственный университет имени М.В. Ломоносова, 119991 Москва, Ленинские горы, д. 1, Россия*

[c]*Московский физико-технический институт, 141700, Россия, Московская область, г. Долгопрудный, Институтский пер., 9*

[d]*НИУ "Высшая школа экономики", 101978 Москва, ул. Мясницкая, д. 20, Россия*

[e]*TigerGraph, 94065 Калифорния, Редвуд-Сити, Твин Дельфин Драйв, 3, США*

[f]*Техкомпания Хуавэй. 121614 Москва, ул. Крылатская, д. 17, к. 2, Россия*

**E-mail: kuzloc@ispras.ru*

***E-mail: pavel.velikhov@tigergraph.com*

****E-mail: fqiang.fuqiang@huawei.com*

Поступила в редакцию 10.09.2022 г.
После доработки 19.09.2022 г.
Принята к публикации 24.09.2022 г.

Аналитика в реальном времени – относительно новая ветвь аналитики. Обычное "определение" аналитики в реальном времени заключается в том, чтобы как можно быстрее анализировать данные по самым последним данным. Это определяет суть фундаментальных потребностей пользователей, но никоим образом не является конкретным требованием к соответствующим программным комплексам в силу нечеткости "определения". В результате разные производители систем управления аналитическими данными и исследователи относят к системам аналитики в реальном времени совершенно разные системы, отличающиеся архитектурой, функциональностью и даже временными параметрами. Цель этой статьи – проанализировать различные подходы к предоставлению аналитики в реальном времени, их преимущества и недостатки, а также компромиссы, на которые неизбежно приходится идти как разработчикам систем, так и их пользователям.

DOI: 10.31857/S0132347423010053, **EDN:** GRYYWT

1. ВВЕДЕНИЕ

Традиционная технология анализа данных (On-Line Analytical Processing, OLAP) в основном базируется на концепции хранилищ данных (Data Warehouse), как она описана, например, в книгах пионеров этого подхода Билла Инмона [1] и Ральфа Кимбалла [2]. Согласно их определениям, хранилище данных – это база данных, в которой тесно интегрируются на физическом уровне данные из разных источников с помощью специальной процедуры извлечения, преобразования, загрузки (Extract, Transform, Load, ETL). Хранилище данных содержит как можно больше исторических данных, чтобы предоставить аналитикам возможность обнаружить с помощью статистики скрытые правила или тенденции, которые могут помочь ответственным лицам принимать правильные *стратегические* бизнес-решения.

21-й век принес новую концепцию аналитики – *анализ данных в реальном времени*. Gartner [3] дает следующее определение: "Аналитика в реальном времени – это дисциплина, которая применяет логику и математику к данным, чтобы предоставить информацию для быстрого принятия лучших решений. В некоторых случаях реальное время просто означает, что аналитика завершается в течение нескольких секунд или минут после поступления новых данных". Как видно, Gartner больше всего выделяет одну важную особенность: анализ данных в реальном времени должен выполняться как можно быстрее. Однако не менее важна и другая особенность аналитики в реаль-

ном времени: данные должны быть максимально свежими.

Существует множество различных вариантов использования аналитики в реальном времени, но все они относятся к *повседневной* деловой деятельности: такой анализ данных помогает оптимизировать рутинные бизнес-операции. В этом случае решения должны приниматься *быстро*, поэтому аналитические операции должны быть *быстрыми,* а данные должны отражать текущую ситуацию в бизнесе, поэтому они должны быть *свежими*. Понятия быстрой аналитической обработки данных и свежести данных не абсолютны, а скорее относительны: программная система, обеспечивающая аналитику в реальном времени, делает все возможное для поддержки быстрых сколь угодно сложных аналитических запросов к свежим данным, однако пользователям всегда придется искать компромисс между их потребностями, преимуществами аналитики в реальном времени и реальные возможностями системы.

Продолжим приведенную выше цитату из Глоссария Gartner: "*Аналитическая система по требованию (on-demand) в реальном времени* ожидает, пока пользователи или системы предоставят запрос, а затем выдают аналитические результаты. *Непрерывная (continuous) аналитика в режиме реального* времени более активна и предупреждает пользователей или инициирует реакцию по мере возникновения событий". В частности, в настоящее время существует два основных подхода к аналитике в реальном времени. Первый подход обеспечивает практически в реальном времени ответы на произвольно сложные аналитические запросы к почти свежим сохраненным данным. Второй предлагает ограниченные аналитические операции в настоящем режиме реального времени над свежими потоковыми данными. По аналогии с общепринятым понятием системы реального времени первый подход можно назвать аналитикой *мягкого (soft)* реального времени, а второй – аналитикой *жесткого (hard)* реального времени.

Первая категория решений, в свою очередь, может быть разделена на два подкласса: так называемые современные хранилища данных (*modern data warehouses*) и системы гибридной транзакционно-аналитической обработки данных (*hybrid transactional/analytical data processing, HTAP*). Системы первого подкласса обычно ориентированы на облачную среду (хотя некоторые поставщики также предоставляют версии своих систем для использования в локальной среде (*on premise*)). Они обеспечивают быструю аналитическую обработку за счет использования массивно-параллельной архитектуры (massively-parallel processing, MPP), хранененения таблиц по столбцам и современных методов оптимизации запросов. Свежесть данных обеспечивается за счет возможности динамического добавления внешних данных, часто в потоковом режиме.

Решения HTAP поддерживают в рамках одной системы как транзакционную, так и аналитическую обработку данных. Оперативные данные, генерируемые транзакциями, становятся почти немедленно доступными для анализа данных. Это общий принцип обеспечения свежести данных в системах HTAP. Быстрая аналитическая обработка запросов обеспечивается за счет хранения данных в основной памяти и распараллеливания запросов на современных многоядерных процессорах.

В мировой литературе имеется множество публикаций, посвященных аналитике в реальном времени, включая, в частности, обзоры (например, [4–7]). Однако, насколько нам известно, в каждой из этих работ рассматриваются технологии и решения, относящиеся к одному конкретному подходу (например, потоковой обработке [4, 5] или HTAP [6, 7]), в предположении, что этот и только этот подход предоставляет аналитику в режиме реального времени.

Напротив, цель нашей статьи состоит в том, чтобы использовать обобщенное понятие анализа данных в (почти) реальном времени, чтобы рассмотреть и представить примеры наиболее успешных реализаций всех соответствующих современных подходов и технологий, которые могут быть использованы в этой области аналитики. В каждом случае мы также предоставим наиболее известные и очевидные варианты использования соответствующего подхода.

Следует отметить, что наш обзор не является исчерпывающим из-за обширности темы и ограниченного размера статьи. Мы также ограничиваем себя, рассматривая только аналитику над обычными структурированными данными реляционной модели данных (точнее, ее воплощения на основе SQL). Важные вопросы аналитики полуструктурированных или неструктурированных данных в режиме реального времени выходят за рамки этой статьи.

Оставшаяся часть статьи имеет следующую структуру. В разд. 2 мы приводим краткую историю потоковой обработки данных (главным образом, анализа), описываем наиболее важные компоненты этой технологии и представляем наиболее известные решения как коммерческих вендоров, так и сообщества открытого кода. В разд. 3 описываются основные функции и архитектурные принципы современных хранилищ данных с упором на облачные решения крупных поставщиков. Мы также анализируем, какие основные изменения в традиционных технологиях хранилищ данных обеспечивают свежесть данных и быструю обработку запросов. Разд. 4 посвящен анализу данных почти в реальном времени

на основе технологии HTAP. Мы рассмотрим корни этой технологии (в основном управление транзакциями в системах, хранящих базы данных в основной памяти, и аналитика с использованием хранения таблиц по столбцам), опишем историю подхода и представим наиболее интересные коммерческие и академические решения с учетом возможностей новой энергонезависимой памяти. Разд. 5 подводит итоги и завершает статью.

2. ПОТОКОВАЯ АНАЛИТИКА ДАННЫХ

Похоже, что понятие потока данных впервые было введено в статье Моники Хензингер (Monika Henzinger), Прабхакара Рагхавана (Prabhakar Raghavan) и Шридара Раджагопалана (Sridar Rajagopalan) [8]. Они определили поток данных как такую последовательность элементов данных $x_1 \dots x_i \dots x_n$, что элементы считываются один раз в порядке возрастания индексов i. Авторы выбрали два параметра – количество проходов по потоку (P) и размер требуемой оперативной памяти (S) (в зависимости от n) – и изучили алгоритмы решения различных задач с малыми значениями P и значениями S, меньшими размера входных данных.

2.1. Первые годы потоковой обработки

Вышеупомянутая статья была скорее теоретической и не предлагала никаких идей анализа потоковых данных, но концепция потоковой передачи данных была настолько жизненно важной, соответствовала стольким важным вариантам использования (например, сенсорным сетям, биржевым рынкам, управлению трафиком и т.д.), что с начала нового века многие исследователи начали работать в этой новой области управления данными – области потоковой обработки данных. Первые результаты этих исследований обсуждались на нескольких встречах в Стэнфордском университете в 2002–2003 гг. [9] и были опубликованы в [10]. На наш взгляд, наиболее важными для будущих исследований и разработок были статьи об Aurora и Medusa [11], TelegraphCQ [12] и Stanford Stream Data Manager [13]. Мы не будем подробно останавливаться на деталях предлагавшихся архитектур и кратко упомянем лишь наиболее важные результаты этих работ.

Мы считаем, что наиболее важной особенностью потоковой аналитики является понятие *непрерывного запроса* (*continuous query*). Эта концепция была введена в 1992 году Дугласом Терри (Douglas Terry) и др. [14], гораздо раньше, чем концепция потоковой передачи данных, и в совершенно другом контексте. Как говорят авторы, “непрерывные запросы аналогичны обычным запросам к базе данных за исключением того, что они выдаются один раз и в дальнейшем выполняются “непрерывно” над базой данных”. Очевидно, что в этом контексте такие запросы имеют смысл тогда и только тогда, когда состояние базы данных изменяется с течением времени, чтобы предоставить релевантные результаты такого запроса в разные моменты времени. Чтобы пользователи могли писать непрерывные запросы, авторы использовали в своей системе Tapestry специальный язык запросов TQL, но во время предварительной обработки запросов система преобразовывала запросы TQL в стандартный SQL.

Другой контекст использования непрерывных запросов был продемонстрирован в проекте NiagaraCQ [15]. Под руководством Дэвида Девитта (David Dewitt) группа из Университета Висконсин-Мэдисон разработала распределенную систему, поддерживающую непрерывные запросы к распределенному набору данных XML. Для формулировки запросов они использовали расширение языка запросов XML-QL, одного из предшественников XQuery. Основной целью этого проекта было обеспечение масштабируемости, достаточной для одновременной поддержки миллионов непрерывных запросов. Для достижения этой цели разработчики пытались группировать похожие запросы и обрабатывать не отдельные запросы, а группы запросов.

Идея непрерывных запросов оказалась очень актуальной в контексте потоковых данных, и эта идея интенсивно использовалась во всех трех проектах, упомянутых в начале этого подраздела и стартовавших почти одновременно в начале 2000-х годов.

2.1.1. TelegraphCQ. Проект TelegraphCQ [12] (более подробное описание можно найти в [16]) был выполнен командой Калифорнийского университета в Беркли, в которую входили Майкл Франклин (Michael Franklin), Джозеф Хеллерштейн (Joseph Hellerstein) и Сэмюэл Мэдден (Samuel Madden). Проект включал в себя несколько этапов прототипирования и редизайна, а TelegraphCQ – это название окончательной архитектуры, изображенной на рис. 1.

Проект родился в Беркли и, естественно, базировался на СУБД PostgreSQL с открытым исходным кодом. Однако для поддержки запросов потоковых данных и достижения других целей проекта команда существенно изменила архитектуру PostgreSQL. Главной из других целей была адаптивность системы. Это мы здесь обсуждать не будем.

Для достижения первой, самой важной цели они в первую очередь адаптируют и расширяют идею *окон* в потоке данных, впервые представленную Йоханнесом Герке (Johannes Gehrke) и др. в [17]. Авторы этой статьи предложили два вида окон – раздвижные (landmark) и скользящие (slading). В раздвижном окне фиксируется старый

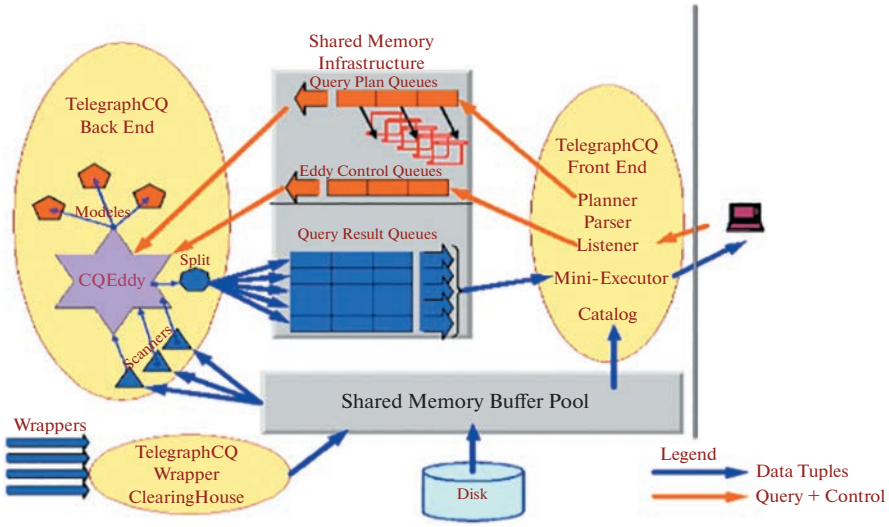

Рис. 1. Архитектура TelegraphCQ.

(левый) конец, а новый (правый) конец перемещается вперед при поступлении в потоке новых кортежей, в то время как в скользящем окне при прибытии новых кортежей оба конца окна перемещаются вперед. В TelegraphCQ использовались гораздо более общие окна, чем раздвижные и скользящее. Была введена специальная конструкция *for-loop* для объявления последовательности окон, в которых пользователь хочет получить ответы на свой запрос: переменная *t* перемещается по временной шкале по мере выполнения цикла, а левый и правый концы каждого окна в последовательности и условие остановки выполнения запроса могут быть определены относительно этой переменной *t*.

Во-вторых, TelegraphCQ обеспечивает возможность одновременного выполнения нескольких запросов, как непрерывных, так и выполняемых однократно, как над потоковыми, так и над историческими (сохраняемыми в базе данных) данными. Для поддержки такой возможности был разработан специальный многопоточный исполнитель запросов, который активно использовал разделяемую память, работал без блокировок и пытался совместно обрабатывать похожие запросы.

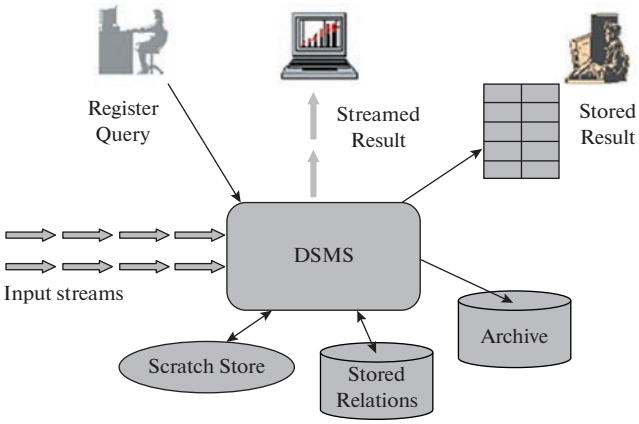

Рис. 2. Высокоуровневое представление STREAM.

Наконец, хотя авторы [12, 16] не акцентируют на этом внимание, но похоже, что они использовали в качестве языка запросов расширенную версию диалекта SQL PostgreSQL.

2.1.2. Stanford Stream Data Manager. Stanford Stream Data Manager (Stream, позже переименованный в Stanford Data Stream Management System) [13] (более подробное описание см. в [18]) естественным образом разрабатывался в Стэнфордском университете при участии (может быть, под руководством) всем известной Дженнифер Видом (Jennifer Widom). Высокоуровневое представление STREAM показано на рис. 2.

Входящие входные потоки бесконечно производят данные и управляют обработкой запросов. Для обработки непрерывных запросов обычно требуется сохранять в основной или внешней памяти промежуточное состояние данных (Scratch Store). Хотя основной задачей Stream являлась оперативная обработка непрерывных запросов, во многих приложениях потоковые данные можно было скопировать в архив для сохранения и возможной автономной обработки аналитических запросов. Пользователи или приложения регистрируют непрерывные запросы, которые остаются активными в системе до тех пор, пока их регистрация явно не будет отменена. Результаты непрерывных запросов обычно передаются в виде выходных потоков данных.

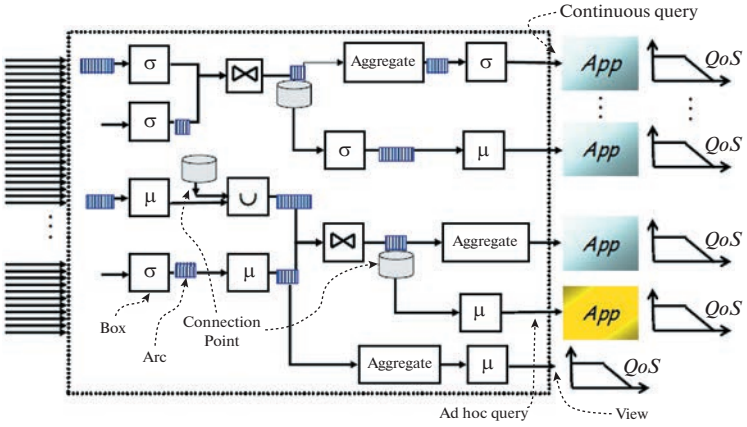

Рис. 3. Сеть обработки в Aurora.

Stream был действительно работающим прототипом системы потоковой обработки, но наиболее важным результатом проекта стал язык CQL (*Continuous Query Language*). Хотя синтаксически CQL является незначительным расширением SQL, его семантика сильно отличается от семантики SQL. В своих статьях о Stream авторы описывают абстрактную семантику SQL, приводят примеры запросов и обсуждают особенности их оптимизации и обработки. Сама команда Stream считала дизайн CQL настолько важным, что представила отдельную длинную статью [19], полностью посвященную возможностям этого языка. Между прочим, эта статья явно демонстрирует, что Дженнифер Видом была основным разработчиком CQL. Язык жив и поддерживается почти во всех современных системах обработки потоков данных.

2.1.3. Aurora, Borealis, StreamBase. Проект Aurora [11] (более подробное описание см. в [20, 21]) был инициирован Майклом Стоунбрейкером, который привлек к участию в проекте профессоров и студентов из MIT, университетов Brown и Brandeis. Отметим, что, насколько нам известно, [11] была первой публикацией, в которой высказывалось предположение, что потоковая обработка данных может служить основой для аналитики в реальном времени. Авторы пишут: “Распространенной целью является разработка технологии “предприятия реального времени”, с помощью которой можно было бы производить бизнес-анализ данных в режиме реального времени. Такой тактический анализ требует перехвата потоков данных из оперативных систем, их объединения и последующей обработки в режиме реального времени. Поддержка предприятий реального времени – одна из целей специализированных систем потоковой обработки, таких как Aurora и Medusa”.

В Aurora непрерывные запросы определялись с помощью визуального графического интерфейса с прямоугольниками и дугами, а систему Aurora можно рассматривать как направленный подграф диаграммы потока работ, который выражает все одновременно выполняемые вычисления результатов запросов. Эта диаграмма потока работ называется сетью Aurora (см. рис. 3). Запросы строятся из стандартного набора четко определенных операторов (блоков). Дуги обозначают очереди кортежей, представляющие потоки. Каждый блок по расписанию обрабатывает один или несколько кортежей из своей входной очереди и помещает результаты в свою выходную очередь. Когда кортежи поступают в очереди, подключенные к приложениям, они оцениваются в соответствии с QoS (quality of service) приложения.

По умолчанию запросы являются непрерывными в том смысле, что они потенциально могут выполняться бесконечно долго для входных данных. Во время работы системы также могут быть определены однократно выполняемые запросы, привязываемые к точкам подключения, которые представляют собой предопределенные дуги в сети, ведущие к хранилищам исторических данных. С точками подключения могут быть связаны спецификации персистентности, которые указывают, как долго должна храниться история. Aurora также позволяет определять обособленные точки подключения, в которых могут храниться статические наборы данных. Наличие точек подключения позволяет направлять запросы к сочетанию традиционных хранимых данных и динамических данных, поступающим в потоках. Aurora также позволяет определять представления, то есть запросы, к которым не подключено ни одно приложение. Представлению разрешается иметь спецификацию QoS для указания на уровень его важности. При наличии потребности приложения могут подключаться к представлениям.

Medusa как отдельный проект упоминается только в [11]. Вкратце, этот проект обеспечил се-

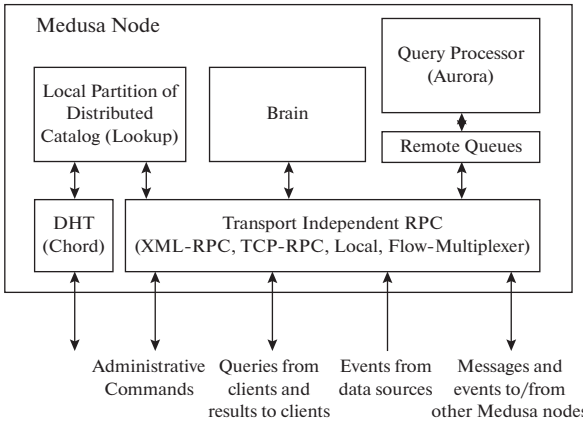

Рис. 4. Высокоуровневая архитектура узла Medusa.

тевую инфраструктуру, которая позволила организовать распределенную обработку потоков на базе Aurora (см. рис. 4).

В конце 2003 года система Aurora была коммерциализирована. Майк Стоунбрейкер и его коллеги основали стартап, который первоначально назывался Grassy Brook, а вскоре был переименован в StreamBase [21]. В то же время межуниверситетская команда в течение следующих трех лет продолжала разработку системы распределенной обработки потоков под названием Borealis [21, 22]. Как показывает рис. 5, архитектура этой системы была фактически объединением Aurora и Medusa.

Компания StreamBase работала самостоятельно десять лет в тесном сотрудничестве с университетами, а затем в 2013 г. была приобретена компанией TIPCO Software, где программное обеспечение StreamBase в настоящее время является основным продуктом для потоковой обработки данных [23]. Вероятно, наиболее интересным вкладом StreamBase в сообщество обработки потоков является язык запросов StreamSQL – диалект SQL, отражающий семантику запросов Aurora [24].

Интересно, что еще в 2008 году Стоугбрейкер инициировал процесс интеграции CQL и StreamSQL с целью определить единый язык потоковых запросов. Эта попытка, результаты которой были опубликованы в [25], не увенчалась успехом, поскольку семантика CQL и StreamSQL слишком различается. Однако обе модели полезны в разных приложениях, поэтому многие современные системы обработки потоков поддерживают оба языка.

Наконец, в результате исследований Майкла Стоунбрейкера и его коллег была опубликована очень разумная, ценная и влиятельная статья [26]. Рекомендации Стоунбрейкера, Четинтемеля (Uğur Çetintemel) и Здоника (Stanley Zdonik) сыграли и продолжают играть важную роль в разработке новых систем потоковой обработки. Мы кратко перечислим здесь эти требования.

1) Система обработки потоковых данных в реальном времени должна обрабатывать сообщения “в потоке”, без необходимости их сохранения для выполнения какой-либо операции или последовательности операций.

2) Система должна поддерживать высокоуровневый язык “StreamSQL” со встроенными расширяемыми примитивами и операторами, ориентированными на потоки.

3) Система должна иметь встроенные механизмы для обеспечения устойчивости к “несовершенствам” потока, включая отсутствующие и неупорядоченные данные.

4) Система должна гарантировать предсказуемые и воспроизводимые результаты.

5) Система должна иметь возможность эффективно хранить, получать доступ и изменять информацию о состоянии, а также объединять ее с потоковыми данными в реальном времени.

6) Чтобы избежать сбоев в обработке данных в реальном времени, система потоковой обработки

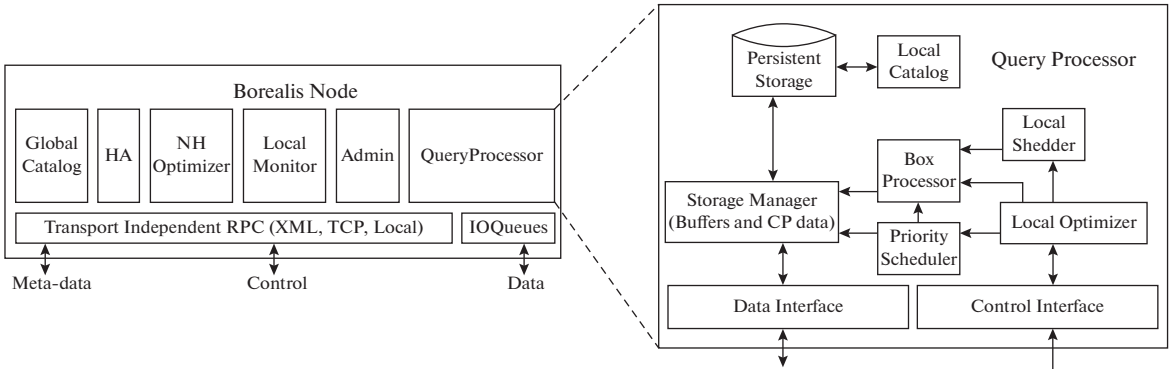

Рис. 5. Архитектура Borealis.

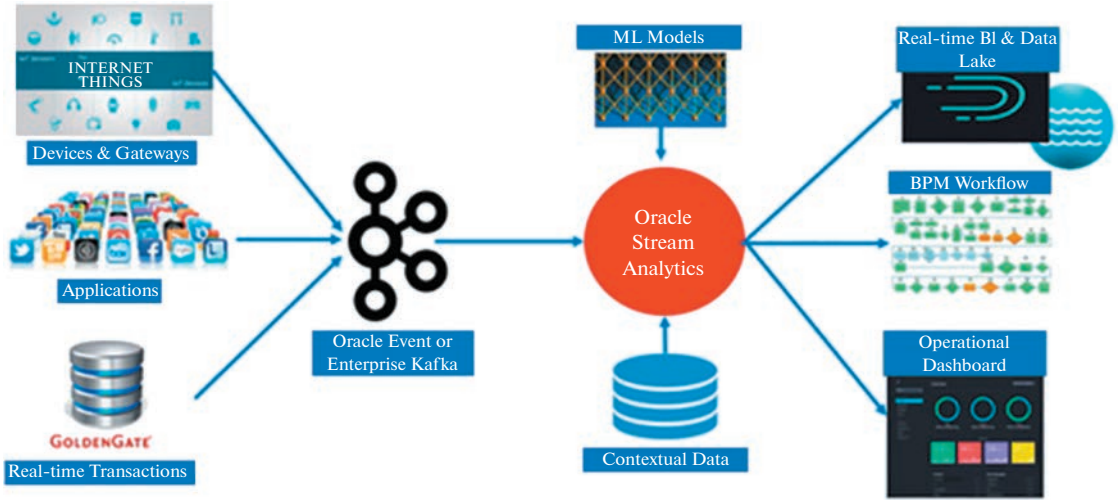

Рис. 6. Архитектура Stream Analytics.

должна использовать решение высокой доступности.

7) Система должна иметь возможность распределять обработку между несколькими процессорами и машинами для достижения дополнительной масштабируемости.

8) Система потоковой обработки должна иметь высокооптимизированный исполнительный механизм с минимальными накладными расходами, чтобы обеспечить отклик в реальном времени для приложений, работающих с большим объемом данных.

2.2. Следующее поколение и современное состояние

В то время как в первое десятилетие 2000-х годов в основном выполнялись фундаментальные исследования принципов потоковой обработки, второе десятилетие включало разработку практически полезных систем, как проприетарных, так и открытых. Общий термин *система управления потоками данных* (*Data Stream Management System, DSMS*), введенный в первое десятилетие, был принят для обозначения любой программной системы, обрабатывающей потоки данных. В то же время появились дополнительные термины *обработка событий* (*Event Processing*), *обработка сложных событий* (*Complex Event Processing, CEP*) или *обработка потока событий* (*Event Stream Processing*) для обозначения систем, специализирующихся на обработке и анализе событий для выявления событий более высокого уровня. DSMS имеют более широкую область применения, и приложения CEP могут быть реализованы с помощью DSMS [27].

Через десять лет после публикации [10] был опубликован специальный выпуск бюллетеня IEEE Data Engineering Bulletin, в котором содержались статьи, описывающие состояние дел в начале 2010-х годов. С наших позиций и в соответствии с современным состоянием технологии основной интерес представляют публикации [29–31]. Мы будем ссылаться на эти публикации в следующих подразделах.

В настоящее время рынок программного обеспечения предлагает множество решений категории DSMS, и здесь мы должны ограничиться лишь несколькими примерами. Эти примеры включают как продукты крупных поставщиков программного обеспечения, так и решения с открытым исходным кодом.

2.2.1. Oracle Stream Analytics. Компания Oracle начала заниматься обработкой потоковых данных много лет назад (см., например, публикацию [32], авторы которой описывают попытку включить обработку потоковых данных и обработку сложных событий в СУБД, первую такую попытку в коммерческих базах данных). В настоящее время основным потоковым продуктом Oracle является Oracle Stream Analytics [33]. Его архитектура изображена на рис. 6.

Oracle Stream Analytics (OSA) – это технология с хранением данных в основной памяти (*in memory*) для аналитических вычислений в реальном времени над потоковыми большими данными. OSA может автоматически обрабатывать и анализировать крупномасштабную информацию в режиме реального времени, используя сложные паттерны корреляции, обогащения данных и алгоритмы машинного обучения. Потоковые большие данные могут исходить от датчиков IoT, вебконвейеров, файлов журналов, устройств торговых точек, банкоматов, социальных сетей, транзакционных баз данных, баз данных NoSQL или любого другого источника данных. OSA работает в масштабируемой и высокодоступной кластер-

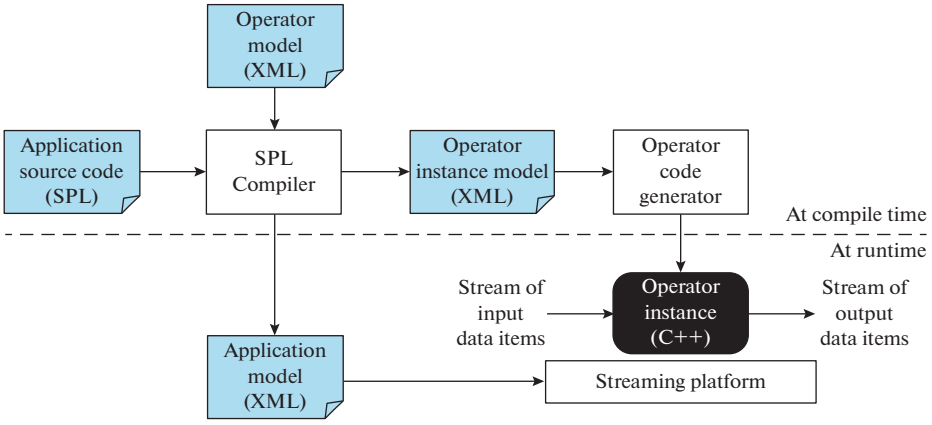

Рис. 7. Компиляция и выполнение приложения, созданного с использованием Streams Processing Language (SPL).

ной среде больших данных с использованием Apache Kafka и Apache Spark Streaming (см. последующие подразделы), интегрированных с Oracle Continuous Query Engine, и обеспечивает решение критических задач в реальном времени на современных предприятиях.

Работа Stream Analytics начинается с приема данных из Kafka с качественной поддержкой сбора данных об изменениях на основе Oracle GoldenGate. Исследование и анализ потока выполняется путем создания конвейеров данных. Конвейер данных может запрашивать данные, используя временны́е окна (с использованием CQL), искать шаблоны и применять условную логику, пока данные все еще находятся в движении. Запросы и конвейер Spark Streaming автоматически генерируются Web-ориентированным инструментальным средством. После анализа данных и обнаружения искомой ситуации конвейер может быть остановлен, чтобы запустить потоки работ BPM (Business Process Modelling) в Oracle Integration или сохранить результаты в Data Lake для обеспечения более глубокого понимания и анализа данных с помощью Oracle Analytics Cloud.

2.2.2. IBM Streams. Система IBM Streams [31, 35] (ранее называвшаяся IBM InfoSphere Streams [36, 37]) – это платформа, которая предоставляет среду программирования (на основе языка SPL – Streams Processing Language) и поддержку времени выполнения для разработки и запуска приложений реального времени над потоковыми данными разных видов (рис. 7).

Сердцем системы, очевидно, является язык обработки потоков. Центральными понятиями SPL являются потоки и операторы. Приложение SPL соответствует потоковому графу. Операторы без входных потоков являются источниками и имеют собственный поток (thread) управления. Большинство других операторов срабатывают только тогда, когда в одном из их входных потоков имеется хотя бы один элемент данных. Точнее говоря, когда в приложении на SPL срабатывает оператор, он выполняет часть кода для обработки элемента входных данных. При активации оператора потребляется один элемент входных данных и может передаваться любое количество элементов данных в выходные потоки. Поток доставляет элементы данных от одного оператора к другому в том порядке, в котором они были отправлены.

Встроенные декларативные операторы SPL поддерживают, в частности, традиционные функции управления окнами, фильтрации и агрегации потоков. Пользователи могут определять свои собственные операторы и использовать их так же, как встроенные. Основное преимущество такого подхода заключается в том, что приложения работают поблизости от поступления данных, и эти приложения относительно безопасны из-за использования одного языка со строгой системой типов.

2.2.3. Потоковая аналитика компании Microsoft. StreamInsight [38–41] – это механизм обработки сложных событий, способный обрабатывать сотни тысяч событий в секунду с чрезвычайно низкой задержкой. Он может быть размещен в любом процессе, таком как служба Windows, или встроен непосредственно в приложение. StreamInsight имеет простую модель адаптера для ввода и вывода данных, а при формулировке запросов как к данным в реальном времени, так и к историческим данным используется один и тот же синтаксис LINQ, доступный так же, как и из любого языка, который поддерживается в Microsoft .NET Framework.

Высокоуровневая архитектура StreamInsight (рис. 8) довольно проста: события собираются из множества источников через входные адаптеры. Эти события анализируются и преобразуются с помощью запросов, а результаты запросов рас-

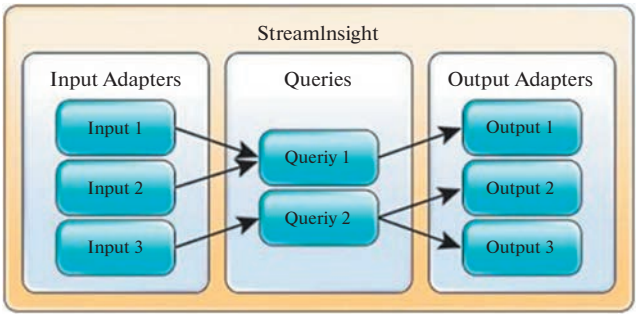

Рис. 8. Высокоуровневая архитектура Microsoft StreamInsight.

пространяются на другие системы и людей через выходные адаптеры.

В StreamInsight в качестве языка запросов используется LINQ, и такие запросы выглядят точно так же, как запросы LINQ to SQL, адресуемые к базе данных. Когда события поступают от адаптера ввода, оценивается их полезность и, если значение свойства Value больше 0.5, они передаются адаптеру вывода.

В настоящее время потоковая аналитика в Microsoft переходит в облачную среду, и наиболее продвигаемым продуктом является Azure Stream Analytics [42]. Azure Stream Analytics (рис. 9) – это механизм аналитики в реальном времени и обработки сложных событий, предназначенный для одновременного анализа и обработки больших объемов быстрых потоковых данных из нескольких источников. Паттерны и взаимосвязи могут обнаруживаться в информации, извлекаемой из ряда источников ввода, включая устройства, датчики, потоки кликов, каналы социальных сетей и приложения. Эти паттерны можно использовать для запуска действий и инициирования потоков работ, таких как создание предупреждений, передача информации в средство формирования отчетов или сохранение преобразованных данных для последующего использования. Кроме того, система Stream Analytics доступна в среде выполнения Azure IoT Edge, что позволяет обрабатывать данные на устройствах IoT.

Задание Azure Stream Analytics состоит из ввода, запроса и вывода. Stream Analytics получает данные из концентраторов событий Azure (Azure Event Hubs), включая концентраторы событий Azure из Apache Kafka, концентратора Интернета вещей Azure (Azure IoT Hub) или хранилища BLOB-объектов Azure (Azure Blob Storage). Запрос, основанный на языке запросов SQL (очевидно, с некоторыми расширениями), можно использовать для простой фильтрации, сортировки, агрегирования и соединения потоковых данных за определенный период времени. SQL может быть дополнительно расширен с помощью пользовательских функций (User-Defined Functions, UDF), написанных на JavaScript и C#. Варианты упорядочения событий и продолжительность временных окон могут быть скорректированы при выполнении операций агрегирования с помощью простых языковых конструкций и/или конфигураций.

Многие другие компании предлагают свои решения для анализа потоковых данных, например, уже упомянутая StreamBase компании TIBCO [23], Data Engineering Streaming компании Informatica [43], Event Stream Processing компании SAS [44], Apama Streaming Analytics компании Software AG и т.д. Мы не будем обсуждать здесь эти продукты и сосредоточимся далее на некоторых чрезвычайно популярных разработках с открытым исходным кодом. Все они принадлежат сообществу Apache.

2.2.4. Apache Kafka and Samza. Эти программные системы традиционно связаны, потому что обе они происходят от LinkedIn, а Samza изначально была построена над Kafka [29]. Авторы [29] называли Kafka брокером сообщений, а Samza – фреймворком для обработки сообщений. Теперь в [45] утверждается, что Kafka является рас-

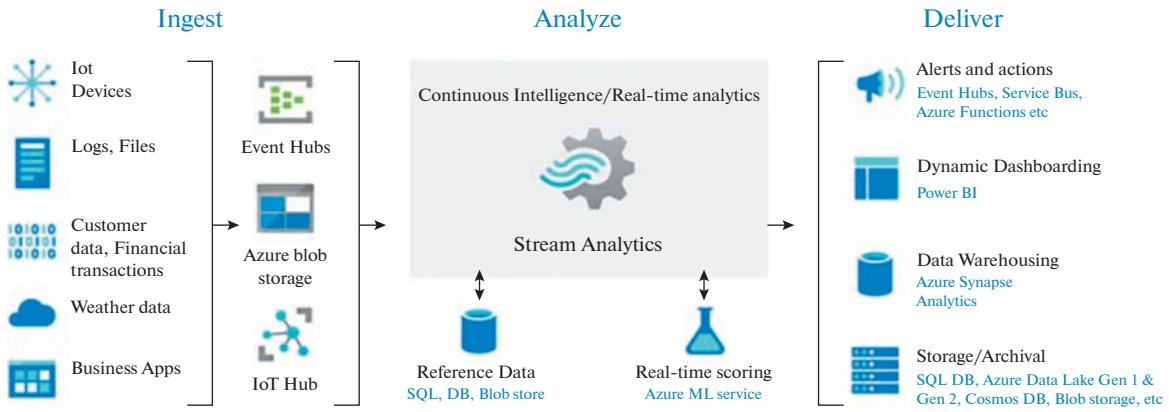

Рис. 9. Конвейер аналитической обработки в Azure Stream Analytics.

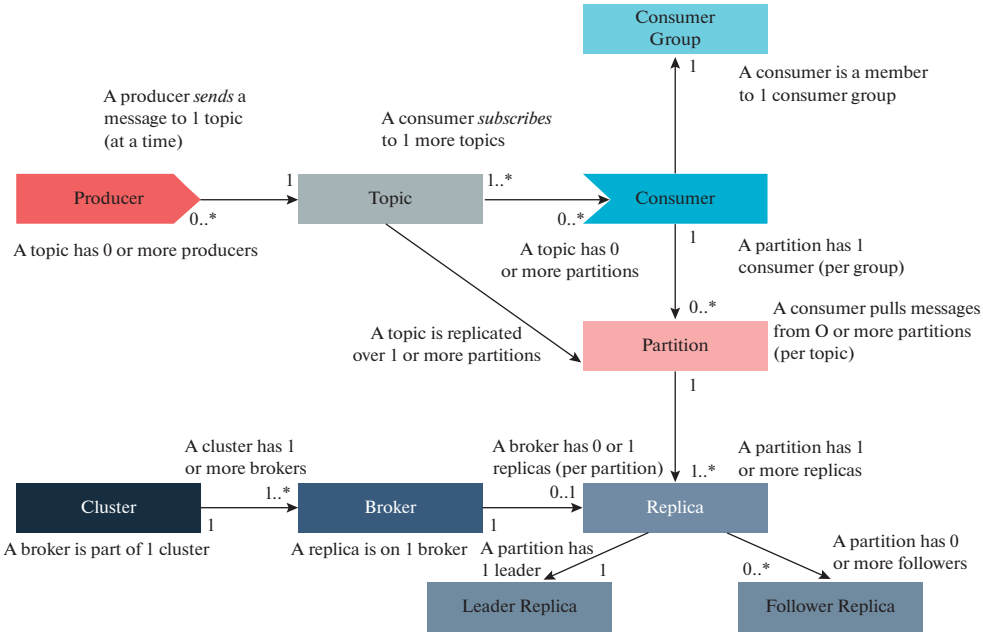

Рис. 10. Основные компоненты Kafka.

пределенной платформой потоковой передачи событий, а [46] называет Samza фреймворком распределенной потоковой обработки.

Так или иначе, Kafka активно используется в различных приложениях и системах для обработки (и анализа) потоков данных. Архитектура и основные компоненты Kafka показаны на рис. 10. Компоненты кратко описываются ниже [47].

• Брокер Kafka – это сервер, работающий в кластере Kafka (кластер Kafka состоит из нескольких брокеров). Как правило, несколько брокеров взаимодействуют, образуя кластер Kafka и добиваясь балансировки нагрузки, надежности на основе избыточности и аварийного переключения. Для управления кластером и координации совместной работы брокеры используют Apache ZooKeeper [48]. Каждый экземпляр брокера способен эффективно обрабатывать сотни тысяч сообщений в секунду суммарного терабайтного объема.

• Производитель (producer) Kafka служит источником данных, который оптимизирует, записывает и публикует сообщения для одной или нескольких тем (topic) Kafka. Производители Kafka также сериализуют, сжимают и балансируют данные между брокерами посредством партиционирования.

• Потребители (consumer) Kafka читают данные, выбирая сообщения из тем, на которые они подписаны. Каждый потребитель принадлежит к некоторой группе потребителей. Каждый потребитель в своей группе потребителей отвечает за чтение части разделов каждой темы, на которую он подписан.

• Тема Kafka определяет канал, по которому передаются данные. Производители публикуют сообщения в темах, а потребители читают сообщения из темы, на которую они подписаны. Темы организуют и структурируют сообщения, причем сообщения заданного типа публикуются в соответствующей теме. Темы в кластере Kafka идентифицируются по уникальным именам, и число создаваемых тем не ограничено.

• В кластере Kafka темы делятся на разделы (partition), и разделы реплицируются между брокерами. Из каждого раздела несколько потребителей могут читать тему параллельно. Также возможно, чтобы производители добавляли к сообщению ключ – все сообщения с одним и тем же ключом отправляются в один и тот же раздел. Сообщения добавляются в разделы последовательно, а сообщения без ключей распределяются по разделам в циклическом режиме.

Архитектура Kafka обеспечивает масштабируемость и высокую производительность, надежность и восстановление после отказов.

Фреймворк Apache Samza (высокоуровневая архитектура изображена на рис. 11) разработан специально для использования преимуществ архитектуры Kafka и гарантирует отказоустойчивость, буферизацию и хранение состояния. Для координации совместного использования ресурсов в Apache Samza используется YARN [49]. Это означает, что по умолчанию требуется кластер

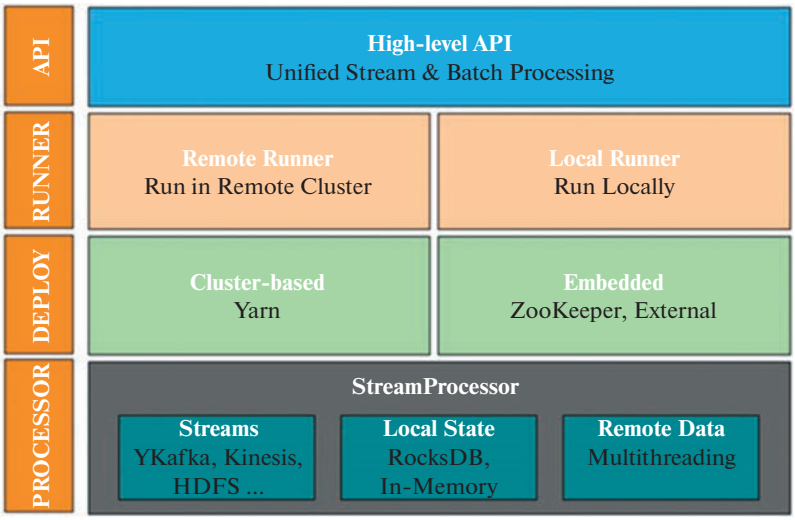

Рис. 11. Высокоуровневая архитектура Apache Samza.

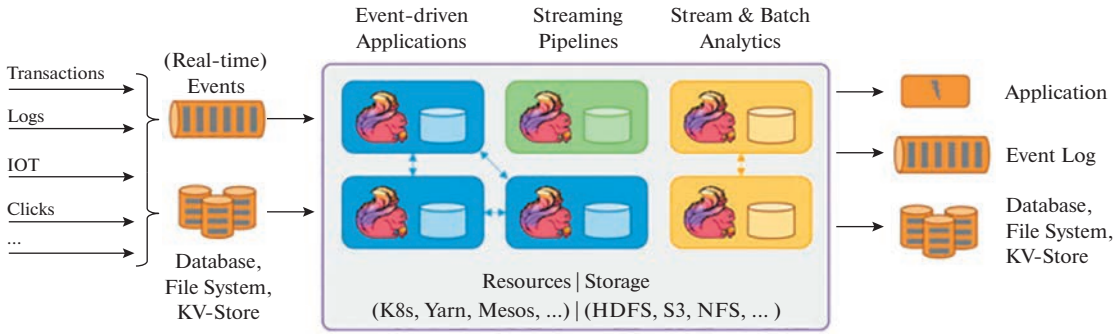

Рис. 12. Высокоуровневое представление Apache Flink.

Hadoop, а Samza полагается на развитые функции, встроенные в YARN.

2.2.5. Apache Flink. Apache Flink [30] – это фреймворк и механизм распределенной обработки для вычислений с отслеживанием состояния над *неограниченными (unbounded)* и *ограниченными (bounded)* потоками данных (см. рис. 12). Flink разработан для использования во всех распространенных кластерных средах, обеспечивая эффективное выполнения вычислений in memory в любом масштабе.

У неограниченных потоков имеется начало, но отсутствует установленный конец. Они не завершаются и предоставляют данные по мере их поступления. Неограниченные потоки должны обрабатываться непрерывно, т.е. события должны обрабатываться сразу же после их приема. У ограниченных потоков имеют установленные начало и конец. Перед выполнением каких-либо вычислений над ограниченным потоком должны быть приняты все его данные. Точный контроль времени и состояния позволяет среде поддержки времени выполнения Flink запускать любые приложения над неограниченными потоками. Ограниченные потоки обрабатываются с применением алгоритмов и структур данных, специально разработанных для наборов данных фиксированного размера, что обеспечивает высокую производительность.

2.2.6. Apache Spark Streaming. Spark Streaming [52] – это расширение базового API Spark [53], которое обеспечивает масштабируемую, высокопроизводительную и отказоустойчивую потоковую обработку потоков данных в реальном времени. Данные могут быть получены из многих источников, таких как сокеты Kafka или TCP (см. рис. 13), и могут быть обработаны с использованием сложных алгоритмов, выраженных с помощью высокоуровневых функций, таких как map, reduce, join и window. Наконец, обработанные данные можно передавать в файловые системы, базы данных и интерактивные информационные панели. К потокам данных можно применять ал-

Рис. 13. Spark Streaming.

горитмы машинного обучения и обработки графов Spark.

Spark Streaming получает потоки входных данных в реальном времени и разделяет данные на пакеты, которые затем обрабатываются механизмом Spark для создания окончательного потока результатов в пакетах. Spark Streaming предоставляет высокоуровневую абстракцию, называемую *дискретизированным потоком*, или *DStream*, которая представляет непрерывный поток данных. DStreams можно создавать либо из входных потоков данных из таких источников, как Kafka и Kinesis, либо путем применения высокоуровневых операций к другим DStreams.

Apache поддерживает ряд других проектов потоковой аналитики, но мы ограничимся четырьмя рассмотренными выше.

2.3. Преимущества, недостатки и компромиссы

Имеются многочисленные варианты использования потоковой аналитики в реальном времени, которые повторяются буквально всеми поставщиками таких решений:

• Обнаружение мошенничества в режиме реального времени на основе алгоритмов машинного обучения обнаружения аномалий.

• Формирование маркетинговых предложений в режиме реального времени на основе местоположения и лояльности клиентов.

• Улучшение обслуживания активов за счет отслеживания нормальных рабочих параметров и упреждающего планирования.

• Повышение доходности за счет постоянного отслеживания спроса и оптимизации уценок вместо случайного снижения цен.

• Корректировка цен путем постоянного отслеживания спроса, уровня запасов и мнений о продуктах в социальных сетях и т.д.

• Повышение уровня продаж продуктов и услуг за счет мгновенного определения присутствия клиента на веб-сайте компании.

• Улучшение использования активов за счет отслеживания среднего времени, необходимого для погрузки и разгрузки товаров.

• Отслеживание операционных событий авиакомпании для исключения задержки рейсов.

• Анализ потоков телеметрии с устройств Интернета Вещей.

• Геопространственная аналитика для управления автопарком и беспилотными транспортными средствами.

• Аналитика в режиме реального времени по данным точек продаж для управления запасами и обнаружения аномалий.

• И так далее.

Во всех этих случаях, когда потоки данных (или событий) генерируются в процессе работы предприятий, эти предприятия получают *реальную выгоду* от потокового анализа.

Потенциальные *недостатки* потоковой аналитики, на наш взгляд, следующие:

• Как мы упоминали во введении, у аналитики в реальном времени имеются два аспекта: она должна обеспечивать *быстрый анализ свежих данных*. Безусловно, любая DSMS *должна* поддерживать настолько быструю обработку запросов, насколько быстры соответствующие потоки данных. В этом смысле эти системы обеспечивают жесткую обработку в реальном времени. Но как насчет свежести данных? DSMS является только потребителем потоковых данных, а не их производителем. По сути, ситуация ничем не отличается от обычной загрузки данных в хранилище (см. разд. 3): непонятно, как проверить, что поступающие данные действительно свежие.

• Пятое требование Стоунбейкера и др. [26] утверждает, что DSMS должна обеспечивать унифицированный доступ на основе интегрированного языка запросов как к потоковым, так и к историческим (хранимым) данным. Это требование вполне разумно с точки зрения аналитика. Однако 5-е требование может привести к значительным проблемам с реализацией. Если объем непрерывного запроса ограничивается только потоковыми данными, то более или менее понятно, как обеспечить его быстрое выполнение: общий объем данных, к которым обращается запрос, ограничен размерами соответствующих окон, и обычно эти данные умещаются в основную память.

Но если запрос обращается как к потоковым, так и к историческим данным, то размер запрашиваемых данных практически неограничен, и единственный способ обеспечить быстрое выполнение таких запросов – ограничить их сложность. Непонятно, как определить такое ограничение, чтобы DSMS оставалась системой жесткого реального времени без чрезмерного снижения возможностей аналитиков.

Необходимые *компромиссы* очевидны: разработчики приложений и аналитики должны четко понимать потенциальные преимущества и ограничения DSMS, тщательно изучать ограничения конкретных систем и не ожидать, что возможно-

сти аналитики в реальном времени будут предоставлены им автоматически.

3. ОБЛАЧНЫЕ ХРАНИЛИЩА ДАННЫХ

Технология облачного хранилища данных относительно нова и постоянно меняется. Общепринятых определений или наборов требований пока нет. Все известные реализации, некоторые из которых кратко обсуждаются ниже, имеют очень разные архитектуры и даже разные цели проектирования, но архитектура этих систем полностью отличается от традиционной архитектуры хранилища данных Инмона (William H. Inmon) [1] и Кимбалла (Ralph Kimball) [2]. Некоторые решения перенимают существующие технологии СУБД, интеграции или преобразования данных и т.д.; другие используют новые методы и методы, но все они основаны на представлении таблиц по столбам как более эффективном подходе к поддержке аналитических запросов.

Наиболее важные общие черты современных облачных хранилищ данных выглядят следующим образом:

• Доступ к данным. Перемещение данных в облако позволяет предоставлять аналитику над данными почти реального времени, получаемыми из различных внешних источников, включая внутренние операционные данные предприятий.

• Производительность оценки запросов. Использование современных методов выполнения аналитических запросов с различными видами распараллеливания на основе передового облачного оборудования обеспечивает производительность, близкую к требованиям реального времени.

• Масштабирование. Облачные хранилища данных масштабируются быстро, а иногда и частично автоматически из-за разъединения вычислений и хранения данных.

Эти черты позволяют называть современные облачные хранилища данных облачными сервисами мягкого реального времени.

В этом разделе мы кратко обсудим архитектуру и основные функции четырех популярных современных облачных хранилищ данных:

• Google BigQuery;

• Amazon RedShift;

• Microsoft Azur Synapse Analytics (dedicated SQL pool) и

• Snowflake.

3.1. Google BigQuery

По словам Google, BigQuery – это “бессерверное, масштабируемое и экономичное мультиоблачное хранилище данных, разработанное для обеспечения адаптивности бизнеса”. [54]. Общая архитектура BigQuery показана на рис. 14.

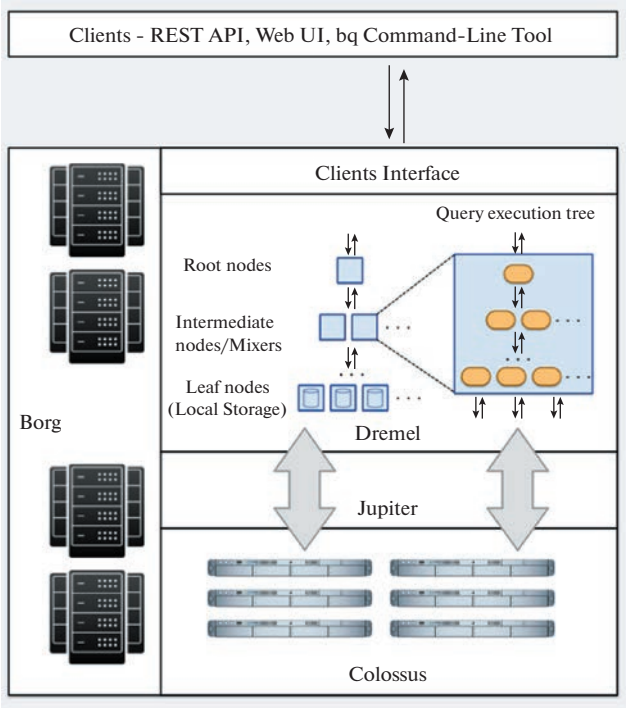

Рис. 14. Высокоуровневая архитектура службы BigQuery.

Основные компоненты BigQuery:

• Dremel – система обработки запросов;

• Borg – система управления кластерами;

• Jupiter – сетевая инфраструктура;

• Colossus – распределенная файловая система.

Dremel [56, 57] – старейший компонент BigQuery, это масштабируемая, интерактивная (быстрая) система обработки произвольных запросов для анализа неизменяемых вложенных данных. Задание Dremel

• считывает данные из файловых систем Google Colossus по сети Jupiter;

• выполняет различные операции SQL и;

• возвращает результаты клиенту.

Dremel использует оригинальное поколоночное представление таблиц Capacitor [58], которое поддерживает вложенные структуры данных и прямые операции над сжатыми данными.

Общая философия обработки запросов в Google такова.

• Никогда не известно, какие запросы потребуются в каждой возможной ситуации.

• Решение BigQuery состоит в том, чтобы увеличить скорость полного сканирования, получая доступ ко всем записям на дисках без индексации или предварительно агрегированных значений.

Поэтому используются многоуровневые деревья обслуживания и тысячи вычислительных узлов для выполнения запросов.

Корневой узел получает входящие запросы, считывает метаданные из таблиц и направляет запросы на следующий уровень дерева обслуживания. Листовые серверы взаимодействуют с уровнем хранения или обращаются к данным на локальном диске.

Схема очень хорошо подходит для простых запросов с агрегацией. При выполнении сложных запросов с агрегацией и других классов запросов используются механизмы, применяемей в параллельных СУБД и Map/Reduce. Детали не ясны, и информация о реализации недоступна.

Colossus [59] обеспечивает репликацию, восстановление и распределенное управление на уровне кластера. Для импорта данных в хранилище BigQuery можно использовать либо пакетную загрузку, либо потоковую передачу. В процессе импорта BigQuery отдельно кодирует каждый столбец в формате Capacitor.

Как только все данные столбца закодированы, они записываются обратно в Colossus. Во время кодирования собирается различная статистика о данных, которая в дальнейшем используется для планирования запросов. При записи данных в Colossus BigQuery принимает решение о первоначальной стратегии партиционирования, которая основывается на паттернов запросов и доступа к данным. После записи данных для обеспечения максимальной доступности BigQuery инициирует георепликацию данных в разных центрах обработки данных. Существующие записи не могут быть обновлены, поэтому BigQuery в основном поддерживает варианты использования только для чтения. Однако всегда можно записать обработанные данные в новые таблицы.

Борг [60] одновременно запускает тысячи заданий Dremel в одном или нескольких кластерах, состоящих из десятков тысяч машин. Кроме того, помимо выделения вычислительных мощностей для заданий Dremel, Borg обеспечивает отказоустойчивость.

Из-за разделения уровней вычислений и хранения данных для BigQuery требуется сверхбыстрая сеть, которая может за секунды доставлять терабайты данных из среды хранения данных в вычислительные узлы для выполнения заданий Dremel. Google Jupiter [61] позволяет сервису BigQuery использовать сети с пропускной способностью до 1-го петабит/с.

3.2. Amazon RedShift

Amazon RedShift основан на PostgreSQL 8.0.2, но исходный код PostgreSQL был сильно изменен [61]. Схема хранения данных и механизм выполнения запросов полностью отличаются от PostgreSQL.

Данные хранятся по столбцах с использованием специальных методов сжатия данных. Redshift автоматически сжимает все данные, которые загружаются в систему, и распаковывает их во время выполнения запроса. Поддержка вторичных индексов и эффективных операций обработки данных в построчном представлении была исключена.

Общая архитектура RedShift изображена на рис. 15 [62, 63]. Кластер – это основной структурный элемент хранилища данных Amazon Redshift, ответственный за обработку запросов. Каждый кластер Redshift состоит из двух основных компонентов.

• Вычислительный узел (Compute Node) имеет собственный центральный процессор, основную память и дисковое хранилище. Вычислительные узлы хранят данные и выполняют запросы, и в одном кластере может быть много узлов.

• Ведущий узел (Leading Node) управляет связью между вычислительными узлами и клиентскими приложениями. Ведущий узел компилирует код, распределяет скомпилированный код по вычислительным узлам и назначает часть данных каждому вычислительному узлу.

Ведущий узел Redshift и вычислительные узлы работают следующим образом. Ведущий узел получает запросы и команды от клиентских программ. Когда клиенты выполняют запрос, ведущий узел отвечает за синтаксический анализ запросов и создание оптимального плана их выполнения на вычислительных узлах над частями данных, хранящимися в каждом узле. На основе плана выполнения ведущий узел создает скомпилированный код и распределяет его по вычислительным узлам для обработки. Наконец, ведущий узел получает и агрегирует результаты и возвращает результаты клиентскому приложению.

Существует два типа узлов: *dense storage node* и *dense compute node*. Объем внешней памяти каждого узла может варьироваться от 160 ГБ до 16 ТБ – использование самого крупного варианта среды хранения данных в узле позволяет хранить в кластере данные петабайтного масштаба. Горячие данные (*hot data*) хранятся на локальных дисках, исторические данные (*historical data*) – в хранилище объектов S3. По мере роста рабочей нагрузки можно увеличивать вычислительную мощность и емкость среды хранения, добавляя узлы в кластер или изменяя тип узла.

Amazon Redshift использует сетевые подключения с высокой пропускной способностью, непосредственную физическую близость узлов и настраиваемые протоколы связи, чтобы обеспечить высокоскоростную сетевую связь между узлами кластера. Вычислительные узлы работают в отдельной изолированной сети, к которой кли-

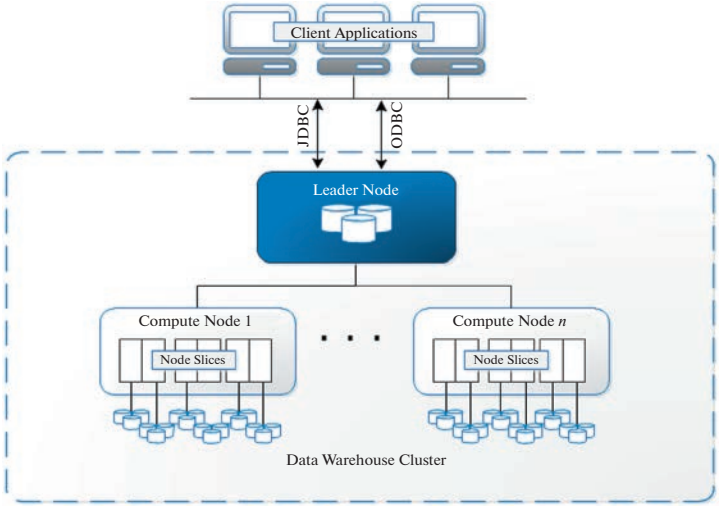

Рис. 15. Высокоуровневая архитектура Amazon Red Shift.

ентские приложения никогда не обращаются напрямую, что обеспечивает дополнительные преимущества в плане безопасности.

Каждый вычислительный узел разбивается на слайсы, и каждый слайс получает часть памяти и дискового пространства. Ведущий узел разделяет данные по слайсам и распределяет между ними части пользовательских запросов или других операций над базой данных. При выполнении операций слайсы работают параллельно. Количество слайсов на узел кластера зависит от размера этого узла кластера.

Внутри узла Redshift может автоматически решать, как распределять данные между слайсами, или можно указать один столбец в качестве ключа распределения. Выполняя запрос, оптимизатор запросов на ведущем узле по мере необходимости перераспределяет данные на вычислительных узлах, чтобы выполнять требуемые операции соединения и агрегации. Предлагаются три “стиля” выбора ключа распределения, чтобы помочь Redshift более эффективно выполнять запросы:

• по общим столбцам разделаются таблица фактов и одна таблицу измерений;

• для разделения выбирается столбец с наибольшим числом различных значений в отфильтрованном частичном результате;

• изменяются некоторые таблицы измерений, чтобы можно было использовать совместное разделение всех таблиц.

Данные, хранящиеся в Redshift, реплицируются на все узлы кластера и автоматически резервируются. Мгновенные снимки сохраняются в S3 и могут быть восстановлены. Amazon отслеживает состояние кластера Redshift, повторно реплицирует данные с отказавших дисков и при необходимости заменяет узлы. Данные могут шифроваться.

Redshift поддерживает SQL-команды UPDATE и DELETE, но не предоставляет команд MERGE или UPSERT для обновления таблицы из одного источника данных. Команда COPY является наиболее эффективным способом загрузки таблицы. Он может читать данные из нескольких файлов данных или нескольких потоков данных одновременно.

3.3. Microsoft Asur Dedicated SQL Poll

В настоящее время Microsoft позиционирует свою современную службу облачного хранилища данных [65, 66] как часть гораздо более крупной службы под названием Azure Synapse Analytics [67] (рис. 16).

Azure Synapse – это корпоративная аналитическая служба, которая ускоряет получение информации из хранилищ данных и систем больших данных. Azure Synapse объединяет технологии SQL, используемые в корпоративных хранилищах данных, технологии Spark, используемые для обработки больших данных, конвейеры для интеграции данных и ETL/ELT, а также обеспечивает глубокую интеграцию с другими службами Azure, такими как Power BI, CosmosDB и AzureML.

В данной работе мы ограничимся рассмотрением компонента Synapse SQL, обеспечивающим функции службы хранилища данных (рис. 17).

Synapse SQL использует масштабируемую архитектуру для распределения вычислительной обработки данных между несколькими узлами. Вычисления отделены от хранилища, что позволяет масштабировать вычисления независимо от данных в системе. Для *выделенного пула SQL* единицей масштабирования является абстракция вычислительной мощности, называемая *data*

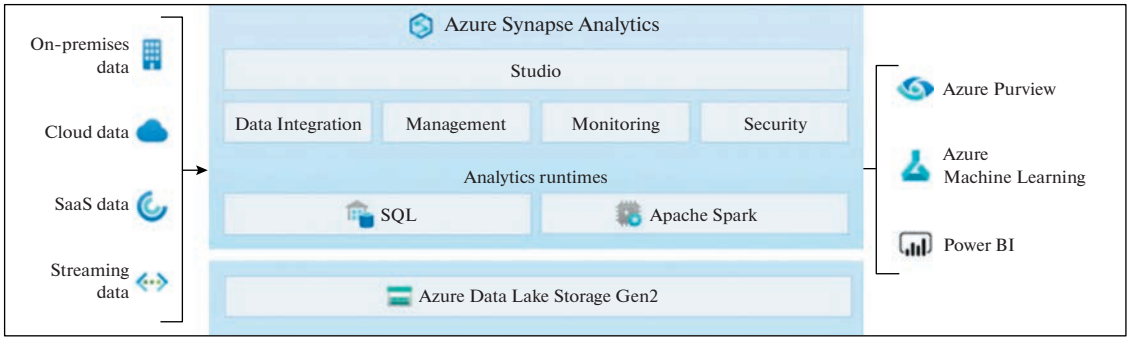

Рис. 16. Azure Synapse Analytics.

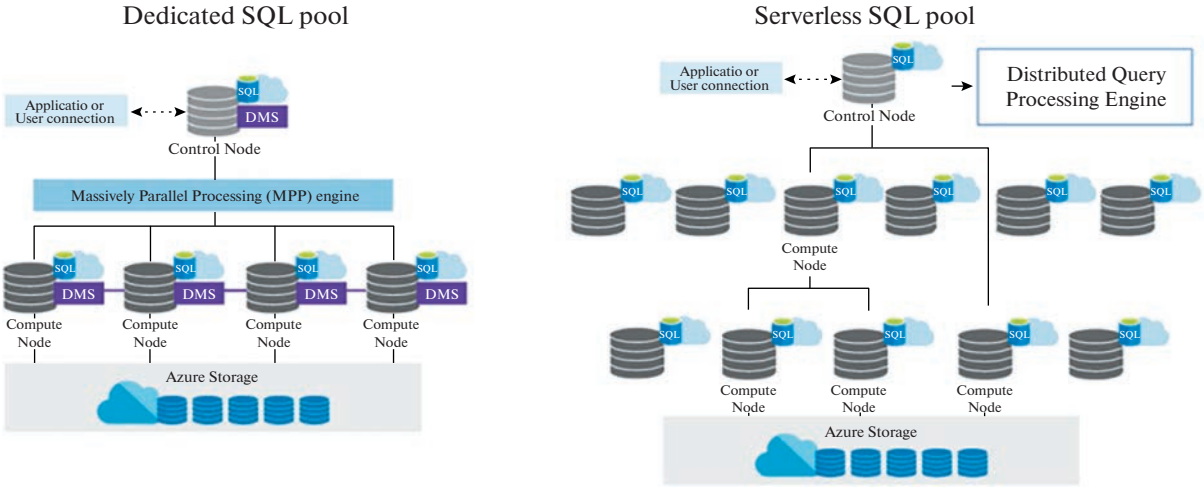

Рис. 17. Архитектура Synapse SQL.

warehouse unit. Масштабирование *бессерверного пула SQL* выполняется автоматически в соответствии с требованиями к ресурсам со стороны запросов. Поскольку топология со временем меняется путем добавления, удаления узлов или аварийного переключения, она адаптируется к изменениям и гарантирует, что запрос имеет достаточно ресурсов и успешно завершается.

Выделенный пул SQL использует архитектуру на основе узлов. Приложения подключаются к *управляющему узлу* и выдают команды T-SQL. На узле управления размещается механизм распределенной обработки запросов, который оптимизирует запросы для параллельной обработки, а затем передает операции вычислительным узлам для параллельного выполнения своей работы. Каждый из нескольких вычислительных узлов содержит экземпляр базы данных SQL и отвечает за обработку данных, хранящихся локально на его дисках. После получения промежуточных результатов вычислительные узлы возвращают эти результаты на управляющий узел для агрегирования.

Выделенный пул SQL сохраняет данные в таблицах с хранением по столбцам. По сравнению с традиционными системами баз данных аналитические запросы выполняются за секунды, а не за минуты или часы. Благодаря разделению средств хранения данных и вычислительной обработки при использовании выделенного пула SQL можно:

- масштабировать вычислительную мощность независимо от потребностей в хранении данных;
- увеличивать или уменьшать вычислительную мощность без перемещения данных;
- приостанавливать использование вычислительных ресурсов, оставляя данные сохранными и платя только за их хранение;
- возобновлять использование вычислительных ресурсов, когда это требуется.

Выделенный пул SQL использует службу хранения данных Azure (хранилище BLOB-объектов Azure) для обеспечения безопасности пользовательских данных. Данные разделяются на дистрибуции (*distribution*) для оптимизации производительности системы. При определении таблицы

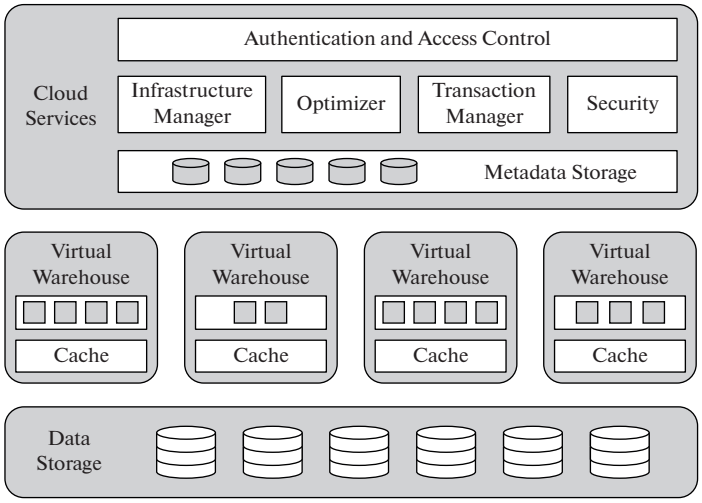

Рис. 18. Архитектура Snowflake.

можно выбирать, какой шаблон будет использоваться для разделения данных. Поддерживаются следующие шаблоны разделения:

- разделение на основе хэширования;
- циклическое разделение;
- разделение с репликацией.

Дистрибуция – это базовая единица хранения и обработки для параллельных запросов, выполняемых над распределенными данными. Когда Synapse SQL выполняет запрос, эта работа делится на 60 более мелких запросов, которые выполняются параллельно. Каждый из 60 мелких запросов выполняется над одной из дистрибуций данных. Каждый вычислительный узел управляет одной или несколькими из 60 дистрибуций. Выделенный пул SQL с максимальным количеством вычислительных ресурсов имеет одно распределение на вычислительный узел. Выделенный пул SQL с минимальными вычислительными ресурсами содержит все дистрибутивы на одном вычислительном узле.

Пул SQL поддерживает транзакции как часть рабочей нагрузки хранилища данных [68]. Однако, чтобы обеспечить поддержку пула SQL в масштабе, некоторые функции ограничены по сравнению с SQL Server:

- Отсутствие распределенных транзакций;
- Не разрешены вложенные транзакции;
- Нет разрешенных точек сохранения;
- Нет именованных транзакций;
- Нет отмеченных транзакций;
- Нет поддержки DDL, такой как CREATE TABLE внутри определяемой пользователем транзакции.

3.4. Snowflake

Snowflake Inc. – самая молодая компания на рынке современных облачных хранилищ данных. Она была основана в 2012 году и уже в 2014 году запустила свой сервис Snowflake [69, 70]. В настоящее время Snowflake является одним из основных конкурентов мировых лидеров рынка. Архитектура Snowflake показана на рис. 18.

- Уровень хранилища данных (Data Storage) использует Amazon S3 для хранения табличных данных и результатов запросов.
- Уровень виртуальных хранилищ (Virtual Warehouse) обрабатывает выполнение запросов в эластичных кластерах виртуальных машин, называемых виртуальными хранилищами.
- Уровень облачных служб (Cloud Services) – это набор служб, которые управляют виртуальными хранилищами, запросами, транзакциями и всеми сопутствующими метаданными: схемами баз данных, информацией об управлении доступом, ключами шифрования, статистикой использования и т.д.

Хранилище данных. Несмотря на то, что производительность S3 может отличаться от случая к случаю, удобство использования этой службы, ее высокую доступность и надежные гарантии долговечности хранения данных трудно превзойти. Было решено потратить основные силы на разработку методов локального кэширования данных и обеспечения устойчивости к перекосам на уровне виртуальных хранилищ. Файлы S3 могут быть (пере)записаны только полностью; невозможно даже добавить данные в конец файла. Однако S3 поддерживает запросы GET для частей файла. Эти свойства оказали сильное влияние на формат файла таблицы Snowflake и схему управления параллелизмом.

Таблицы горизонтально разделяются на большие неизменяемые файлы, которые являются эквивалентами блоков или страниц в традиционной системе баз данных. Внутри каждого файла значения каждого атрибута, или столбца сгруппированы вместе и сильно сжаты с использованием схемы PAX [71]. Каждый файл таблицы имеет заголовок, который, среди прочих метаданных, содержит смещения каждого столбца в файле. Поскольку S3 разрешает запросы GET к частям файлов, исполнителям запросов нужно загружать только заголовки файлов и те столбцы, которые их интересуют.

S3 также используется для хранения временных данных, сгенерированных исполнителями запросов, после исчерпания места на локальном диске, а также больших результатов запросов. Перенос временных данных в S3 позволяет системе выполнять произвольно сложные запросы без проявления ошибок нехватки основной или дисковой памяти.

Хранение результатов запросов в S3 позволяет использовать новые формы взаимодействия с клиентами и упрощает обработку запросов, поскольку устраняет необходимость в курсорах на стороне сервера. Метаданные – объекты каталога, статистика, блокировки и т.д. – хранятся в масштабируемом транзакционном хранилище "ключ-значение", которое является частью уровня облачных служб.

Виртуальные хранилища. Уровень виртуальных хранилищ состоит из кластеров экземпляров EC2. Каждый такой кластер предоставляется единственному пользователю через абстракцию, называемую виртуальным хранилищем (VW). Отдельные экземпляры EC2, составляющие VW, называются рабочими узлами. Пользователи не знают и не заботятся о том, какие или сколько рабочих узлов составляют VW. Вместо этого характеристики VW представлены абстрактными "размерами футболок" от X-Small до XX-Large.

VW – это чистые вычислительные ресурсы. Их можно создавать, уничтожать или изменять размер в любой момент по требованию. Каждый отдельный запрос выполняется ровно на одном VW. Рабочие узлы не используются совместно несколькими виртуальными машинами, что приводит к строгой изоляции производительности выполнения запросов. Когда поступает новый запрос, каждый рабочий узел в соответствующем VW порождает новый рабочий процесс. Каждый рабочий процесс существует только на время выполнения своего запроса. У каждого пользователя в любой момент времени может быть запущено несколько виртуальных машин, и каждая виртуальная машина, в свою очередь, может одновременно выполнять несколько запросов. Каждый VW имеет доступ к одним и тем же таблицам без необходимости физического копирования данных.

Каждый рабочий узел поддерживает кэш табличных данных на локальном диске. Кэш – это набор файлов таблиц, к которым в прошлом обращался узел, точнее, заголовки файлов и отдельные столбцы файлов, поскольку исполнители запросов загружают только те столбцы, которые им нужны. Чтобы улучшить частоту успешных обращений к кэшу и избежать избыточного кэширования отдельных файлов таблиц на рабочих узлах VW, оптимизатор запросов назначает наборы входных файлов рабочим узлам, используя консистентное хэширование (consistent hashing) [72] имен файлов таблиц.

Обработка перекосов также особенно важна в облачном хранилище данных. Некоторые узлы могут работать намного медленнее других из-за проблем с виртуализацией или конфликтов в сети. Помимо прочего, Snowflake решает эту проблему на уровне сканирования:

• всякий раз, когда рабочий процесс завершает сканирование своего набора входных файлов, он запрашивает дополнительные файлы у других узлов того же VW (перехват – stealing);

• если при поступлении такой заявки рабочий узел обнаруживает, что в его наборе файлов осталось много необработанных файлов, он отвечает на эту заявку, передавая запросившему узлу право собственности на один оставшийся файл на время выполнения текущего запроса;

• затем запросивший узел загружает файл непосредственно из S3, а не из кэша прежнего владельца файла.

Основные характеристики *исполнителя запросов SQL*: использование поколоночного представления таблиц, векторизованное исполнение и применение техники проталкивания (push). Векторизованное исполнение означает, что Snowflake избегает материализации промежуточных результатов; данные обрабатываются конвейерным способом, пакетами по несколько тысяч строк в поколоночном формате. Исполнение на основе проталкивания означает, что реляционные операции передают свои результаты нижестоящим операциям, а не ждут, пока эти операции сами извлекут данные (как в модели в стиле Volcano). Это позволяет повысить эффективность использования кэша и обрабатывать планы выполнения запросов, представляемые в виде ориентированных графов без циклов, а не просто деревьев.

В Snowflake отсутствуют многие источники накладных расходов, возникающих при традиционной обработке запросов. Нет необходимости в управлении транзакциями во время выполнения; запросы выполняются над фиксированным набором неизменяемых файлов. Нет буферного пула в

основной памяти. Большинство запросов сканируют большие объемы данных. Использование основной памяти для буферизации таблиц, а не для выполнения операций в данном случае является плохим вариантом. Однако Snowflake позволяет всем основным операциям (объединение, группировка, сортировка) переносить данные на диск и обратно при исчерпании основной памяти. Исполнитель запросов, работающий полностью а основной памяти, хотя и компактнее и, возможно, быстрее, слишком ограничен для поддержки всех интересных рабочих нагрузок.

Облачные сервисы. Слой облачных сервисов является мультитенантным. Каждая служба этого уровня – управление доступом, оптимизатор запросов, менеджер транзакций и другие – является долгоживущей и совместно используемой многими пользователями. Каждая служба реплицируется для обеспечения высокой доступности и масштабируемости. Поэтому сбой отдельных сервисных узлов не приводит к потере данных или доступности, хотя некоторые выполняемые запросы могут завершаться с ошибкой (и быть пригодными для повторного выполнения).

Управление запросами и оптимизация. Все запросы, выдаваемые пользователями, проходят через уровень облачных служб. Обрабатываются все ранние этапы жизненного цикла запроса: синтаксический анализ, контроль доступа и оптимизация плана выполнения запроса. Оптимизатор запросов следует типичному подходу в стиле Cascades [73] с нисходящей оптимизацией на основе оценки стоимости. Вся статистика, используемая для оптимизации, автоматически поддерживается при загрузке и обновлении данных. Поскольку Snowflake не использует индексы, пространство поиска плана меньше, чем в некоторых других системах. Пространство плана еще больше сокращается за счет откладывания многих решений до времени выполнения, например, определение типа распределения данных при выполнении соединений. Такой подход уменьшает количество неправильных решений, принимаемых оптимизатором, повышая надежность за счет небольшой потери пиковой производительности. После завершения работы оптимизатора результирующий план выполнения распространяется на все рабочие узлы, являющиеся частью запроса.

Параллельное управление. В аналитических рабочих нагрузках, как правило, преобладают крупные операции чтения, массовые или незначительные вставки и массовые обновления. ACID-транзакции поддерживаются с помощью изоляции мгновенных снимков (Snapshot Isolation, SI): все операции чтения внутри транзакции видят согласованный мгновенный снимок базы данных ко времени начала транзакции. SI реализуется поверх многоверсионного контроля параллелизма (multi-version concurrency control, MVCC), что означает, что копия каждого измененного объекта базы данных сохраняется в течение некоторого времени. Операции записи (вставка, обновление, удаление, слияние) в таблице создают более новую версию таблицы путем добавления и удаления целых файлов в предыдущей версии таблицы. Эти мгновенные снимки также используются для реализации переходов по времени и эффективного клонирования объектов базы данных.

Отсечение (pruning) против индексации. Поддержка индексов может быть сложным, дорогостоящим и рискованным процессом. Произвольный доступ является проблемой как из-за среды хранения данных (S3), так и из-за формата данных (сжатые файлы); поддержка индексов значительно увеличивает объем данных и время загрузки данных; пользователям необходимо явно создавать индексы, что сильно противоречит сервисному подходу. Альтернативным методом является использование отсечений на основе минимального и максимального значений, карт зон (zone map) и пропуск данных (data skipping). Система поддерживает информацию о распределении данных для заданного фрагмента данных (набор записей, файл, блок и т.д.), в частности, минимальные и максимальные значения в чанке.

3.5. Преимущества, ограничения и компромиссы

Несмотря на различные архитектурные и реализационные подходы, в рассмотренных системах имеются несколько важных общих черт, которые позволяют им обеспечивать быструю (почти в реальном времени) аналитику на достаточно свежих (почти в реальном времени) данных. К общим *выгодным* для пользователей особенностям относятся:

- поколоночный сжатый формат для хранения таблиц и работы с ними;
- массивно-параллельная обработка запросов с логически не разделенными ресурсами между узлами;
- логическое горизонтальное разделение базы данных между вычислительными узлами;
- физическое разделение вычислительных ресурсов и ресурсов хранения с физической возможностью для любого вычислительного узла получить доступ ко всем данным в общем хранилище данных;
- раздельное масштабирование вычислительной части и системы хранения;
- надежная поддержка использования локального хранилища вычислительных узлов для индивидуального кэширования физически общих данных;
- некоторый вид контроля параллелизма, который позволяет, по крайней мере, оперативно

добавлять новые данные в массовом порядке или небольшими порциями.

Однако, как и в случае DSMS, свежесть данных сильно зависит от качества источников данных, а скорость обработки запросов сильно зависит от сложности конкретного запроса. Ответ в течение одной секунды не может быть гарантирован для любого запроса. Таким образом, свойство обработки в мягком реальном времени в основном условно. Это очевидные *ограничения*.

Поэтому разработчики приложений и аналитики не должны слепо верить в то, что современные облачные хранилища данных работают в режиме мягкого реального времени. Они должны трезво оценивать свои реальные потребности и возможности конкретных облачных сервисов.

4. ГИБРИДНАЯ ТРАНЗАКЦИОННО-АНАЛИТИЧЕСКАЯ ОБРАБОТКА ДАННЫХ

Наиболее естественным источником свежих корпоративных данных являются данные, генерируемые транзакциями одного и того же предприятия. Обычно все корпоративные транзакции обрабатываются какой-либо СУБД, ориентированной на OLTP, а корпоративная аналитика поддерживается какой-либо СУБД, ориентированной на OLAP. Для обеспечения аналитики в реальном времени необходимо очень быстро передавать данные из хранилища транзакционной системы в хранилище аналитической системы. Другими словами, для обеспечения аналитики свежих транзакционных данных необходимо обеспечить очень быстрый механизм ETL. Но никто не знает, как реализовать такой быстрый ETL, и чтобы сделать возможной аналитику в реальном времени, была введена концепция гибридной транзакционно-аналитической обработки (HTAP).

“Идеалистическая” цель HTAP-подхода – одновременно обеспечить в рамках одной системы и функциональность, и производительность специализированных OLTP- и OLAP-СУБД – представляется недостижимой. Прагматическая цель этого подхода – построить систему, которая обеспечивает приемлемую пропускную способность для транзакционных рабочих нагрузок и одновременно аналитику в режиме реального времени на достаточно свежих данных – несомненно достижима.

Требования разумной пропускной способности для транзакционных рабочих нагрузок и актуальности данных для аналитических запросов явно противоречат друг другу. Для обеспечения максимальной актуальности данных необходимо сделать доступными для анализа все данные, генерируемые выполняемыми в данный момент транзакциями. Однако тогда транзакции будут конкурировать с аналитическими запросами, и обработка транзакций станет медленнее. Все известные системы HTAP используют тот или иной компромисс, чтобы разумно (в некоторой степени) удовлетворить эти противоречивые требования путем разделения аналитической и транзакционной частей базы данных и, возможно, преобразования данных из представления, хорошо подходящего для обработки транзакций (как правило, построчного формата хранения таблиц) в представление, более подходящее для аналитики (обычно поколоночный формат хранения таблиц).

Еще одним аспектом являются среда хранения данных. Большинство современных СУБД категории HTAP оптимизированы для хранения данных в основной памяти. Это означает, что все структуры данных и алгоритмы, используемые в таких системах, разработаны с учетом того, что вся обработка данных будет выполняться в основной памяти без ввода/вывода с внешними запоминающими устройствами. Вообще говоря, концепция in-memory СУБД не нова. Согласно [74], первая попытка реализовать систему баз данных, хранящую все данные в памяти, была предпринята IBM в 1976 г. (IMS Fast Path). Новая волна in-memory СУБД наблюдалась в 1990-х годах. Тогда и позже (в начале 2000-х) такие СУБД в основном специализировались на обработке транзакций. Сейчас почти все существующие HTAP-СУБД в той или иной степени относятся к классу in-memory систем. Хранение всех данных (или, по крайней мере, *горячих* данных) в основной памяти обеспечивает более высокую скорость обработки транзакций, что частично сглаживает негативные последствия параллельно выполняемых аналитических запросов.

В этом разделе мы кратко обсудим происхождение и основные архитектурные особенности некоторых систем категории HTAP. Более подробный обзор см., например, в [7].

4.1. SAP HANA

HANA, аббревиаиура от *High-Performance Analytic Appliance*, является основным продуктом управления базами данных компании SAP SE. Прежде всего, система используется в составе ERP-решений самой SAP. Однако HANA набирает все большую популярность как отдельный инструмент для разработки и поддержки приложений баз данных, которым требуется как транзакционная, так и аналитическая обработка данных. Система доступна в различных воплощениях, таких как кластерная или облачная версии. Однако здесь мы ограничиваемся основной конфигурацией HANA на одном компьютере.

HANA не разрабатывалась с нуля. На дизайн исполнительной подсистемы HANA с поколо-

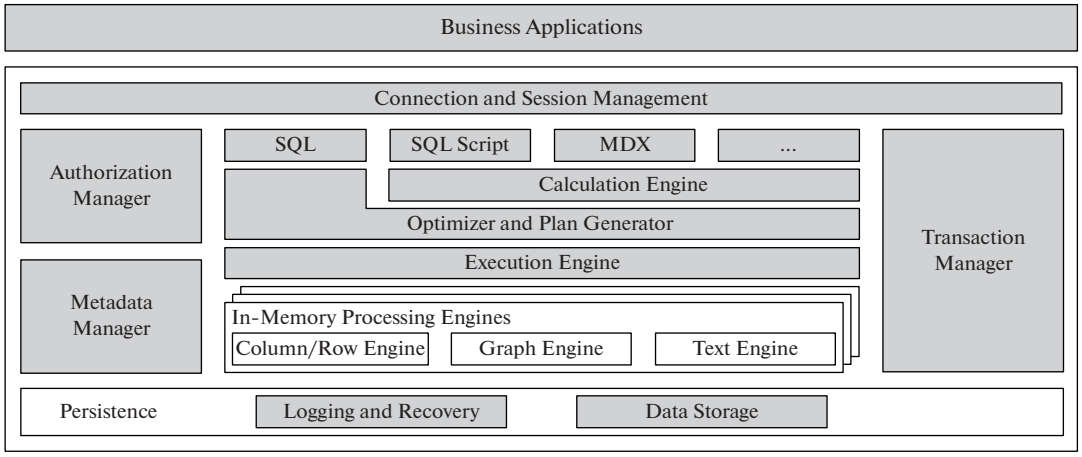

Рис. 19. Архитектура HANA.

ночным хранением таблиц в основной памяти повлияли собственные разработки SAP, система текстового поиска TREX (2001 г.) [75] и SAP Business Warehouse Accelerator (BWA, 2005 г.) [76]. Исполнительная подсистема с хранением таблиц по строкам в основной памяти основана на решении P*TIME [77], разработанном компанией Transact in Memory, Inc., которое было приобретено SAP в 2005 г. Механизм MaxDB [78], приобретенный SAP в 1997 г. у Software AG (позже называвшийся Adabas D) использовался для обеспечения уровня персистентного хранения данных. Первый вариант HANA был выпущен SAP в конце 2010 года.

Рис. 19 демонстрирует общую архитектуру HANA [79]. Мы обсуждаем только часть этой архитектуры, связанную с механизмами HTAP в основной памяти.

HANA позволяет использовать для одной и той же базы данных как хранилища таблиц по столбцам, так и хранилища по строкам. При создании новой таблицы пользователь может выбрать способ ее хранения: по строкам или по столбцам. Согласно исходным проектным решениям [16], поколоночное хранилище является предпочтительным, а HANA оптимизирована для такого типа организации таблиц. Вероятно, использование поколоночного хранилища не позволяет системе продемонстрировать максимальную производительность обработки транзакций, поскольку обновление хранилища по столбцам – это трудоемкая операция. Однако взамен колоночный способ хранения базы данных снижает затраты памяти за счет сжатия данных. Чтобы достичь более высокой скорости обработки транзакций, можно использовать дорогостоящее хранилище строк.

Как правило, каждый столбец таблицы хранится в хранилище столбцов с использованием двух структур данных – словаря и индексного вектора. Словарь содержит все различные значения, находящиеся в данный момент в столбце, а индексный вектор связывает каждую строку таблицы с соответствующей записью словаря. Для поддержки распространенных в аналитике запросов в диапазоне значений HANA сохраняет отсортированный порядок записей словаря. Однако это свойство делает очень дорогими операции обновления, часто используемые при обработке транзакций. Поэтому для каждого столбца поколоночной таблицы имеются две части хранилища: основная часть хранит словарь столбца в отсортированном порядке, а в дельта-части этот порядок не поддерживается. Все операции над таблицей используют как основные данные каждого столбца, так и его дельту. Основываясь на информации о размере дельты и текущей рабочей нагрузки, система решает, когда следует выполнить слияние основной части данных столбца с его дельтой. Во время выполнения этой операции используется новая дельта.

SAP рекомендует использовать хранилище строк для небольших таблиц, в которых строки часто обновляются или удаляются. Таблицы, хранящиеся по строкам, всегда сохраняются в основной памяти. Однако таблицы, которые хранятся по столбцам, могут быть очень большими, и они могут быть разбиты на два раздела, один из которых хранится в памяти, а другой – на дисках. SAP называет раздел таблицы в основной памяти текущим, а раздел на диске – историческим (другие поставщики вместо этого используют термины “горячий” и “холодный” соответственно). В некотором смысле текущая часть таблицы должна содержать данные, которые активно используются в данный момент. HANA может автоматически распределять данные между этими разделами в зависимости от времени жизни определенных строк таблицы или с использованием явных подсказок, предоставляемых приложениями. Теку-

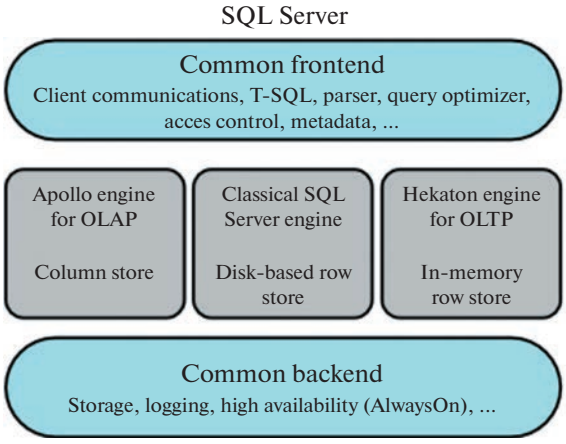

Рис. 20. Упрощенная архитектура SQL Server 2014.

щие и исторические разделы имеют разные схемы хранения, но доступ к ним можно получить с помощью одного запроса.

Имеются несколько дополнительных архитектурных решений, позволяющих одновременно и эффективно обрабатывать транзакции и аналитические запросы в одной системе. Во-первых, каждый аналитический запрос компилируется и оптимизируется с учетом самой свежей статистики базы данных. Запросы, используемые в повторяющихся (параметризованных) транзакциях, готовятся заранее; при обработке транзакций используются готовые исполняемые планы запросов. Во-вторых, аналитические запросы выполняются на высоком уровне параллелизма с использованием нескольких потоков, работающих на разных ядрах ЦП. Распараллеливание транзакционных запросов считается вредным: переключение контекста обходится слишком дорого. Поэтому каждый транзакционный (простой) запрос полностью выполняется в рамках одного потока.

Наконец, чтобы свести к минимуму негативные, потенциально замедляющие обработку транзакций эффекты конкуренции между операциями транзакций и аналитическими запросами, транзакционные операции всегда имеют более высокий приоритет, чем запросы OLAP.

4.2. Аналитика реального времени в основной памяти компании Microsoft

Похоже, компания Microsoft внедрила в SQL Server этот вид аналитики реального времени в три этапа. Во-первых, Microsoft разработала поколоночное хранилище на диске [80]. Затем была разработана и реализована система OLTP в основной памяти Hekaton [81]. И, наконец, компания разработала аналитическую систему для своего хранилища по столбцам и интегрировала все эти компоненты в SQL Server 2014 (и усовершенствовала полученный результат в SQL Server 2016 и SQL Server 2019), обеспечив тем самым аналитику в реальном времени в стиле HTAP [82].

Упрощенная архитектура SQL Server 2014 изображена на рис. 20.

Аналитику в реальном времени обеспечивают компоненты Hekaton и Apollo. Hekaton – это специализированная транзакционная подсистема, использующая хранилище таблиц по строкам в основной памяти. Apollo – это специализированная аналитическая подсистема, использующая хранилище таблиц по столбцам в дисковой памяти. Собственно, Apollo – это не название какого-либо продукта, а всего лишь условное название проекта. Оперативные данные очень быстро передаются из Hekaton в Apollo и преобразуются из представления по строкам в поколоночный формат. (Последнюю операцию можно рассматривать как разновидность облегченного ETL.)

Основные архитектурные идеи Hekaton заключаются в следующем. Индексы (хэш и своего рода B-Tree) разработаны и оптимизированы для данных, постоянно хранимых в основной памяти. Операции с индексами не журнализуются, и все индексы перестраиваются во время восстановления базы данных. Все внутренние структуры данных – описатели свободной и занятой памяти, хэш-индексы и индексы диапазонов, а также карта транзакций – полностью свободны от блокировок. Операции SQL и хранимые процедуры компилируются в настраиваемый высокоэффективный машинный код.

Система идентифицирует горячие и холодные данные и сохраняет в основной памяти только горячие данные [83]. Основная идея заключается в том, что не все данные транзакционной базы данных должны быть доступны одинаково быстро; часть из них может быть размещена во внешнем хранилище без замедления обработки транзакций. Hekaton использует LRU-подобный алгоритм, чтобы определить, какие данные активно не используются транзакциями, и перемещает соответствующие части таблиц в дисковое хранилище. Разумеется, в памяти и на дисках используются разные физические схемы расположения строк. Представление на основе столбцов для целей аналитики поддерживается для обеих частей таблиц.

4.3. Oracle Database In-Memory

In-memory решение Oracle основано на in-memory СУБД TimesTen [84]. Проект TimesTen был основан в Hewlett-Packard Laboratories Мари-Энн Неймат (Marie-Anne Neimat). В то время система называлась SmallDB [85] и предназначалась для использования во внутренних целях HP. В 1996 году Неймат основала новый стартап TimesTen Inc.

С этого времени TimesTen активно использовалась в качестве транзакционной СУБД реального времени во многих приложениях, особенно в сфере телекоммуникаций.

В 2005 году TimesTen была приобретена Oracle, интегрирована с общей инфраструктурой Oracle и начала использоваться в качестве кэша основной СУБД Oracle. Интегрированная опция Oracle Database In-Memory, поддерживающая функциональность HTAP, впервые появилась в Oracle 12c в 2013 году [86].

Система поддерживает два типа хранилищ данных: хранилище по строкам находится на дисках и хранилище по столбцам – в памяти. Можно выборочно назначить важные таблицы или их разделы резидентными в хранилище по столбцам, используя при этом буферный кеш для остальной части таблицы. Итак, есть две основные области памяти для размещения частей базы данных: традиционный кеш для блоков внешней памяти и хранилище по столбцам в основной памяти.

Части одной и той же таблицы, хранящиеся в разных представлениях (и в разных хранилищах), обновляются синхронно.

Хранилище по строкам обеспечивает быстрый транзакционный доступ к данным с помощью тщательно разработанных индексов и кэширования блоков в памяти. Хранилище по столбцам оптимизировано для обработки аналитических запросов. Свежесть данных для аналитики обеспечивается автоматической синхронизацией содержимого хранилищ по столбцам и по строкам при вставке, удалении или обновлении строк. Многоверсионность, используемая в управлении транзакциями, позволяет снизить конкуренцию за данные между транзакциями и аналитическими запросами.

Oracle Database In-Memory может быть развернута в инфраструктуре Oracle Real Application Cluster (RAC) [87]. Общая архитектура показана на рис. 21.

В этом случае доступ к блокам данных, содержащим части хранилища строк, и их изменение осуществляется через общий буферный кэш базы данных. Кроме того, каждый отдельный экземпляр базы данных можно настроить на использование частного хранилища по столбцам столбцов в основной памяти. Все таблицы горизонтально разделяются по всем узлам кластера.

Если система работает в режиме in-memory, Exadata Database Machine автоматически переформатирует все данные, к которым осуществляется доступ, в поколоночный формат в основной памяти и сохраняет их в так называемом flash-кэше (внешнее хранилище большого объема на основе flash-памяти) [88]. После этого все части запросов, выгруженные в Exadata, выполняются так же, как если бы они обрабатывались в памяти в узле RAC. В частности, Exadata использует инструкции SIMD для обработки столбцов таблицы.

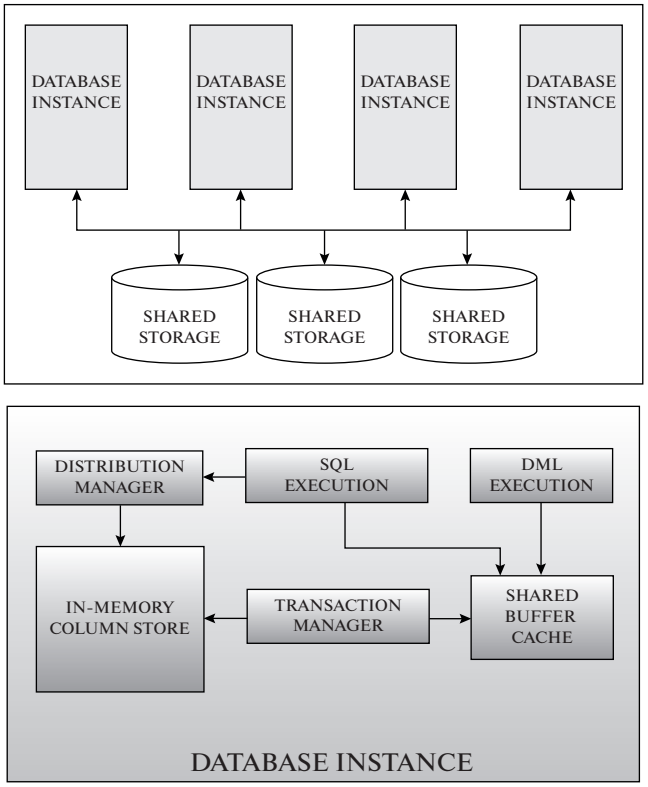

Рис. 21. Архитектура Database In-Memory в Oracle RAC.

4.4. DB2 with BLU Acceleration

В IBM DB2 ускорение аналитики осуществляется с использованием поколоночного хранилища в основной памяти, многоядерного распараллеливания и аппаратной векторной обработки. В 2006 г. началась работа IBM над собственной поколоночной подсистемой в основной памяти. Первым in-memory продуктом, разработанным командой Гая Лохмана (Guy Lohman), был Blink [89]. Это в чистом виде подсистема хранения столбцов в основной памяти. Она успешно используется и сейчас, например, в Informix Warehouse Accelerator [90]. BLU [91] – это продукт второго поколения. Общая архитектура DB2 с BLU показана на рис. 22 [92].

В базе данных, управляемой DB2 с BLU, каждая таблица может храниться либо в традиционном хранилище по строкам, либо в хранилище BLU по столбцам BLU. Важной особенностью BLU является то, что таблицы в поколоночном хранилище могут быть больше, чем размер доступной основной памяти. Фактически все таблицы хранятся в блоках внешней памяти и доступны через обычный блочный кэш.

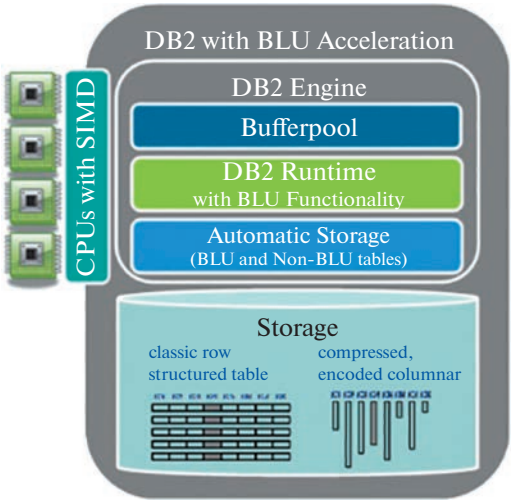

Рис. 22. Архитектура DB2 with BLU.

Однако BLU пытается полностью обработать каждый запрос в основной памяти (хотя технически в этом нет необходимости) и обеспечивает соответствующее управление рабочей нагрузкой. Чтобы обеспечить доступность всех необходимых данных в памяти, BLU использует агрессивную политику предварительной выборки.

Другие особенности DB2 с BLU кажутся традиционными для современных in-memory СУБД, хотя разработчики BLU подчеркивают очень высокую эффективность их реализации. Система поддерживает возможность сканирования и сравнения сжатых данных. Разработчики избегают блокировок, чтобы обеспечить максимальное распараллеливание. Все структуры данных спроектированы таким образом, чтобы свести к минимуму промахи кэша данных и кэша инструкций. Активно используются SIMD-инструкции.

4.4. HyPer

Проект HyPer реализуется командой базы данных Мюнхенского технического университета под руководством Альфонса Кемпера и Томаса Ноймана. Хотя это университетский проект, он не имеет открытого исходного кода. Исходный код системы никогда не публиковался. Более того, в 2015 году был создан околоуниверситетский стартап HyPer, а в 2016 году его приобрела компания Tableau Software, которая называет свою версию СУБД Hyper [93]. Мы не знаем подробностей этой сделки, но каким-то образом развитие HyPer в Мюнхенском университете продолжается. Первый известный отчет о проекте [94] был опубликован в 2010 г.

HyPer – настоящая СУБД в оперативной памяти для многоядерных компьютеров. Система поддерживает как строковые, так и поколоночные хранилища. Начальный формат таблицы может быть выбран при создании таблицы, а физическая структура этой таблицы может меняться в зависимости от рабочей нагрузки.

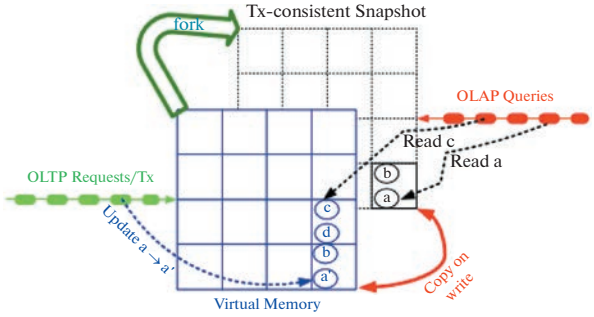

Рис. 23. Мгновенные снимки виртуальной памяти.

Первая версия архитектуры HyPer была очень простой и элегантной, хотя и имела некоторые ограничения. Основные идеи заключались в следующем [95].

Предполагалось, что большинство транзакций могут быть подготовлены заранее (pre-canned), содержать лишь несколько простых запросов и обращаться только к нескольким кортежам. Такие транзакции называются короткими. Все короткие транзакции выполняются последовательно в основном OLTP-процессе. Вся основная область памяти, в которой находится база данных, отображается в виртуальную память этого процесса.

Процесс OLTP периодически (раз в несколько секунд) разветвляет процесс OLAP после фиксации какой-либо только что завершенной транзакции. После этого процесс OLTP включает механизм копирования при записи, который предоставляет новую страницу основной памяти для любой первой операции записи в соответствующую страницу виртуальной памяти. Таким образом, вновь созданный OLAP-процесс видит в своей виртуальной памяти согласованный образ базы данных (согласованный мгновенный снимок). Процесс OLAP выполняет аналитические запросы над этим мгновенным снимком, свежесть которого определяется периодом создания новых процессов OLAP (см. рис. 23).

Транзакция, которая выполняет слишком много запросов или обращается к слишком большому количеству кортежей, считается длинной. Обнаруживаемая выполняемая длинная транзакция откатывается и перенаправляется в текущий процесс OLAP, который имитирует ее выполнение на текущем согласованном мгновенном снимке. Затем эта (частично измененная) транзакция снова перенаправляется в общую очередь транзакций как короткая транзакция. При ее выполнении система сначала проверяет все смоделиро-

ванные обновления по фактическому состоянию базы данных, а затем, если эта проверка проходит успешно, выполняет все эти обновления.

Аналитические запросы распараллеливаются по всем доступным ядрам с использованием массивно-параллельной версии известного алгоритма соединения сортировка-слияние [96].

Позже разработчики HyPer изменили эту простую архитектуру, чтобы обеспечить более высокую пропускную способность для транзакционных рабочих нагрузок. В частности, они внедрили в систему собственную версию многоверсионного контроля параллелизма [96], что значительно усложняет систему, но повышает ее транзакционную производительность.

4.5. Преимущества, недостатки и компромиссы

Как видно из приведенного выше обзора, существует множество способов реализовать функциональность HTAP и обеспечить аналитику в реальном времени. Общие принципы заключаются в следующем:

• предоставлять свежие данные для анализа, объединяя транзакционную и аналитическую обработку в одной системе баз данных;

• сделать обработку транзакций максимально быстрой, чтобы удовлетворить потребности транзакционных клиентов и повысить актуальность данных;

• максимально быстро сделать оперативные данные доступными для анализа, чтобы удовлетворить первое требование аналитики в реальном времени – свежесть данных;

• сделать выполнение аналитических запросов максимально быстрым, чтобы удовлетворить второе требование аналитики в реальном времени – быстрый анализ данных.

Обычный подход для достижения этих целей состоит в том, чтобы хранить всю базу данных в основной памяти. Этот выбор позволяет практически избежать любого ввода-вывода с внешними устройствами хранения (постоянное хранилище используется только для обеспечения долговечности результатов транзакций) и использовать прямые указатели на объекты базы данных внутри базы данных. Все внутренние структуры данных проектируются с учетом наличия аппаратного кэша. Для дальнейшего повышения производительности используются передовые методы оптимизации запросов, компиляция запросов в собственный машинный код, многоядерное распараллеливание, инструкции SIMD и аппаратные ускорители, такие как GPU и FPGA.

Общее *преимущество* подхода HTAP заключается в том, что аналитикам требуется только одна СУБД, чтобы получить возможности быстрого анализа свежих операционных данных.

Главный *недостаток* заключается в том, что практически невозможно обеспечить в одной системе наилучшие характеристики транзакционных и аналитических СУБД.

Обычно одна СУБД поддерживает два хранилища данных – по строкам и по столбцам. Этот подход представляет собой *компромисс* между возможностью быстрой обработки транзакций (хранилище по строкам) и быстрой обработкой аналитических запросов (хранилище по столбцам), с одной стороны, и необходимостью преобразования данных из построчного формата в поколоночный формат до того, как данные станут доступны для аналитики, с другой стороны.

Чтобы избежать высокого уровня конкуренции между параллельно выполняемыми транзакциями, а также между транзакциями и параллельно выполняемыми аналитическими запросами, обычно используется какой-либо многоверсионный контроль параллелизма, часто оптимистичный. Все внутренние объекты базы данных, такие как индексы, распределители памяти и т.д., обрабатываются без блокировок.

Однако хранение базы данных полностью в памяти также является *компромиссом* между высокой производительностью СУБД и ограниченным размером базы данных. Чтобы снять это ограничение, крупные поставщики любят комбинировать хранилища в памяти и на диске. Утверждают, что после этого производительность СУБД не падает (непонятно почему), но в любом случае архитектура системы значительно усложняется.

Имеются некоторые попытки предоставить функции HTAP и аналитики в реальном времени в массивно-параллельных архитектурах без общих ресурсов. Мы считаем, что такая цель является труднодостижимой. Во-первых, даже если каждый узел такой системы будет хранить базу данных целиком в памяти, неизбежные сетевые коммуникации значительно снизят производительность системы. Во-вторых, как транзакционные, так и аналитические СУБД без совместного использования полагаются на подходящее разделение данных между узлами системы. Однако цели партиционирования различны для транзакционных и аналитических СУБД. Транзакционная СУБД пытается разделить базу данных, чтобы минимизировать количество распределенных транзакций. Аналитическая СУБД при партиционировании базы данных старается минимизировать количество распределенных соединений. Сомнительно, чтобы можно было достичь обеих целей в одной системе.

Наконец, значительное число поставщиков in-memory HTAP-СУБД уже используют доступную в настоящее время энергонезависимую основную память (Non-Volatile Memory, NVM) в своих продуктах и/или планируют использовать ее в буду-

щем [98–100]. Эти продукты демонстрируют достаточно широкий спектр вариантов использования NVM от простого расширения памяти с помощью NVM до наиболее перспективных одноуровневых архитектур хранения. Однако последний вариант использования в настоящее время (частично) реализован только в исследовательских проектах [101, 102].

Причина, вероятно, в том, что в настоящее время на рынке доступен только один вид NVM – Intel Optane на базе технологии PCM. Эти модули DIMM на основе NVM имеют довольно высокую задержку и могут не заменить DRAM. Мы считаем, что широко ожидаемое новое поколение NVM (вероятно, основанное на STT) значительно ускорит внедрение NVM в HTAP и (чисто транзакционных) СУБД.

5. ЗАКЛЮЧЕНИЕ

За последние два десятилетия технологии баз данных сделали гигантский скачок вперед. Одним из наиболее заметных достижений является новая возможность анализа данных в реальном времени. Однако этот термин является относительно новым и обычно применяется к программным средам с разными целями и реальными возможностями.

В этой статье мы рассмотрели три подхода к организации управления данными – потоковая обработка, облачное хранилище данных и гибридная транзакционная/аналитическая обработка – каждый из которых претендует на обеспечение аналитики в реальном времени. Мы показали общие архитектурные принципы каждого подхода, привели примеры соответствующих программных продуктов, а также кратко описали их основные преимущества, ограничения и компромиссы.

БЛАГОДАРНОСТИ

Статья подготовлена по материалам доклада на Седьмой Международной конференции “Актуальные проблемы системной и программной инженерии” (АПСПИ 2021).

СПИСОК ЛИТЕРАТУРЫ

1. *William H. Inmon*. Building the Data Warehouse. John Wiley & Sons, 1992. 312 p.
2. *Ralph Kimball*. The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Data Warehouses. Wiley, 1996. 374 p.
3. Information Technology. Gartner Glossary. Real-time Analytics. Available at https://www.gartner.com/en/information-technology/glossary/real-time-analytics, accessed: 06/16/2021
4. *Arun Kejariwal, Sanjeev Kulkarni, Karthik Ramasamy*. Real Time Analytics: Algorithms and Systems. Extended version of VLDB’15 tutorial proposal. arXiv:1708.02621, 2017. 7 p.
5. *Zoran Milosevic, Weisi Chen, Andrew Berry, Fethi A. Rabhi*. Real-Time Analytics. In Big Data: Principles and Paradigms, Morgan Kaufmann, 2016. P. 39–61.
6. *Fatma Özcan, Yuanyuan Tian, Pınar Tözün*. Hybrid Transactional/Analytical Processing: A Survey. Proceedings of the 2017 ACM International Conference on Management of Data, 2017. P. 1771–1775.
7. *Кузнецов С.Д., Велихов П.Е., Фу Ц.* Аналитика в реальном времени, гибридная транзакционная/аналитическая обработка, управление данными в основной памяти и энергонезависимая память. Труды ИСП РАН. 2021. Т. 33. Вып. 3. С. 171–198. https://doi.org/10.15514/ISPRAS–2021–33(3)–13 / Sergey D. Kuznetsov, Pavel E. Velikhov, and Qiang Fu. Real-time analytics, hybrid transactional/analytical processing, in-memory data management, and non-volatile memory. Proceedings of the Ivannikov ISPRAS Open Conference, 2020. P. 78–90.
8. *Monika Rauch Henzinger, Prabhakar Raghavan, and Sridar Rajagopalan*. Computing on data streams. SRC Technical Note 1998-11, May 26, 1998. 16 p.
9. The “Stream Team” Page. Available at http://infolab.stanford.edu/sdt/, accessed 07.07.2021.
10. Special Issue on Data Stream Processing. IEEE Bulletin of the Technical Committee on Data Engineering. 2003. V. 26. № 1.
11. *Stan Zdonik, Michael Stonebraker et al.* The Aurora and Medusa Projects. IEEE Bulletin of the Technical Committee on Data Engineering. 2003 V. 26. № 1. P. 3–10.
12. *Sailesh Krishnamurthy, Sirish Chandrasekaran et al.* TelegraphCQ: An Architectural Status Report. IEEE Bulletin of the Technical Committee on Data Engineering. 2003. V. 26. № 1. P. 11–18.
13. *Arasu A., Babcock B. et al.* STREAM: The Stanford Stream Data Manager. IEEE Bulletin of the Technical Committee on Data Engineering. 2003. V. 26. № 1. P. 19–26.
14. *Douglas Terry, David Goldberg, David Nichols, Brian Oki*. Continuous queries over append-only databases. ACM SIGMOD Record. 1992. V. 21. Iss. 2. P. 321–330.
15. *Jianjun Chen, David J. DeWitt, Feng Tian, Yuan Wang.* NiagaraCQ: A Scalable Continuous Query System for Internet Databases. ACM SIGMOD Record. 2000. V. 29. Iss. 2. P. 379–390.
16. *Sirish Chandrasekaran, Owen Cooper et al.* TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. Proceedings of the 2003 CIDR Conference, 2003. 12 p.
17. *Johannes Gehrke, Flip Korn, Divesh Srivastava*. On Computing Correlated Aggregates Over Continual Data Streams. Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data. 2001. P. 13–24.
18. *Arvind Arasu, Brian Babcock et al.* STREAM: The Stanford Data Stream Management System. Technical Report. Stanford InfoLab, 2004. 21 p. Later appeared as a chapter in Data Stream Management. Processing High-Speed Data Streams. Springer, 2016. P. 317–336.

19. *Arvind Arasu, Shivnath Babu, Jennifer Widom*. CQL: A Language for Continuous Queries over Streams and Relations. Lecture Notes in Computer Science. 2003. V. 2921. P. 1–19.
20. *Daniel J. Abadi*. Don Carney, et al. Aurora: a new model and architecture for data stream management. The International Journal on Very Large Data Bases. 2003. V. 12. Iss. 2. P. 120–139.
21. *Uğur Çetintemel, Daniel Abadi*. The Aurora and Borealis Stream Processing Engines. A chapter in Data Stream Management. Processing High-Speed Data Streams. Springer, 2016. P. 337–359.
22. *Daniel J. Abadi, Yanif Ahmad et al.* The Design of the Borealis Stream Processing Engine. Proceedings of the 2005 CIDR Conference, 2005. P. 277–289.
23. TIBCO StreamBase. Available at https://www.tibco.com/sites/tibco/files/resources/DS-TIBCO-StreamBase-final.pdf, accessed 07/14/2021.
24. StreamSQL Guide. Available at https://docs.tibco.com/pub/sb-lv/2.1.8/doc/html/streamsql/index.html, accessed 07/14/2021.
25. *Namit Jain, Shailendra Mishra et al.* Towards a Streaming SQL Standard. Proceedings of the VLDB Endowment. 2008. V. 1. Iss. 2. P. 1379–1390.
26. *Michael Stonebraker, Uğur Çetintemel, Stan Zdonik*. The 8 requirements of real-time stream processing. ACM SIGMOD Record. 2005. V. 34. Iss. 4. P. 42–47.
27. *Sandra Geisler*. Data Stream Management Systems. In Data Exchange, Integration, and Streams. Dagstuhl Follow-Ups. 2013. V. 5. P. 275–304.
28. Special Issue on Next-Generation Stream Processing. IEEE Bulletin of the Technical Committee on Data Engineering. 2013. V. 38. № 4.
29. *Martin Kleppmann, Jay Kreps*. Kafka, Samza and the Unix Philosophy of Distributed Data. IEEE Bulletin of the Technical Committee on Data Engineering. 2013. V. 38. № 4. P. 4–14.
30. *Paris Carbone, Stephan Ewen*. Apache Flink: Stream and Batch Processing in a Single Engine. IEEE Bulletin of the Technical Committee on Data Engineering. 2013. V. 38. № 4. P. 28–38.
31. *Scott Schneider, Buğra Gedik, Martin Hirzel*. Language Runtime and Optimizations in IBM Streams. IEEE Bulletin of the Technical Committee on Data Engineering. 2013. V. 38. № 4. P. 61–72.
32. *Andrew Witkowski, Srikanth Bellamkonda et al.* Continuous Queries in Oracle. Proceedings of the 33rd International Conference on Very Large Data Bases, 2007. P. 1173–1184.
33. Oracle Fusion Middleware Understanding Stream Analytics. Available at https://docs.oracle.com/en/middleware/fusion-middleware/osa/18.1/understanding-stream-analytics/understanding-oracle-stream-analytics.pdf, accessed 07/16/2021.
34. *Thomas Vengal*. What is Oracle Stream Analytics? Available at https://blogs.oracle.com/dataintegration/what-is-oracle-stream-analytics, accessed 07/16/2021.
35. IBM Streams. Available at https://www.ibm.com/cloud/streaming-analytics, accessed 07/16/2021.
36. *Alain Biem, Eric Bouillet et al.* IBM InfoSphere Streams for Scalable, Real-Time, Intelligent Transportation Services. Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data, 2010. P. 1093–1104.
37. *Hirzel M., Andrade H. et al.* IBM Streams Processing Language: Analyzing BigData in motion. IBM Journal of Research and Development. 2013. V. 57. № 3/4. 11 p.
38. *Mohamed Ali, Badrish Chandramouli et al.* Spatio-Temporal Stream Processing in Microsoft StreamInsight. IEEE Bulletin of the Technical Committee on Data Engineering. 2010. V. 33. № 2. P. 69–74.
39. M*ohamed Ali, Badrish Chandramouli* et al. The Extensibility Framework in Microsoft StreamInsight. Proceedings of the IEEE 27th International Conference on Data Engineering, 2011. P. 1242–1253.
40. *Rob Pierry*. StreamInsight – Master Large Data Streams with Microsoft StreamInsight. MSDN Magazine. 2011. V. 26. № 06.
41. What is Microsoft StreamInsight? Available at https://azurecloudai.blog/2013/01/30/what-is-microsoft-streaminsight/, accessed 07/16/2021.
42. Welcome to Azure Stream Analytics. Available at https://docs.microsoft.com/en-us/azure/stream-analytics/stream-analytics-introduction, accessed 07/16/2021.
43. Data Engineering Streaming. Available at https://www.informatica.com/products/big-data/big-data-streaming.html, accessed 07/16/2021.
44. SAS's Event Stream Processing. Available at https://www.sas.com/en_us/software/event-stream-processing.html, accessed 07/16/2021.
45. Apache Kafka. Available at https://kafka.apache.org/, accessed 07/16/2021.
46. Apache Samza. Available at http://samza.apache.org/, accessed 07/16/2021.
47. Apache Kafka Architecture – Kafka Component Overview. Available at https://www.instaclustr.com/apache-kafka-architecture/#, accessed 07/16/2021.
48. Apache ZooKeeper. Available at https://zookeeper.apache.org/, accessed 07/16/2021.
49. Apache Hadoop YARN. Available at https://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html, accessed 07/16/2021.
50. *Rahul Anand*. What is Apache Samza? Available at https://www.quora.com/What-is-Apache-Samza-1, accessed 07/16/2021.
51. What is Apache Flink? – Architecture. Available at https://flink.apache.org/flink-architecture.html, accessed 07/16/2021.
52. Spark Streaming Programming Guide. Available at https://spark.apache.org/docs/latest/streaming-programming-guide.html, accessed 07/16/2021.
53. Spark API Documentation. Available at https://spark.apache.org/docs/2.4.0/api.html, accessed 07/16/2021.
54. BigQuery. Available at https://cloud.google.com/bigquery, accessed 07/17/2021.
55. A Deep Dive into Google BigQuery Architecture. Available at

https://panoply.io/data-warehouse-guide/bigquery-architecture/, accessed 07/17/2021.

56. *Sergey Melnik, Andrey Gubarev et al.* Dremel: Interactive Analysis of Web-Scale Datasets. Proceedings of the VLDB Endowment. 2010. V. 3. № 1. P. 330–339.
57. *Foto N. Afrati, Dan Delorey et al.* Storing and Querying Tree Structured Records in Dremel. Proceedings of the VLDB Endowment. 2014. V. 7. № 11. P. 1131–1142.
58. *Mosha Pasumansky*. Inside Capacitor, BigQuery's next-generation columnar storage format. Available at https://cloud.google.com/blog/products/bigquery/inside-capacitor-bigquerys-next-generation-columnar-storage-format, accessed 07/17/2021.
59. *Dean Hildebrand, Denis Serenyi*. Colossus under the hood: a peek into Google's scalable storage system. Available at https://cloud.google.com/blog/products/storage-data-transfer/a-peek-behind-colossus-googles-file-system, accessed 07/17/2021.
60. *Abhishek Verma, Luis Pedrosa et al*. Large-scale cluster management at Google with Borg. Proceedings of the Tenth European Conference on Computer Systems, 2015. P. 1–17.
61. *Arjun Singh, Joon Ong, et al.* Jupiter Rising: A Decade of Clos Topologies and Centralized Control in Google's Datacenter Network. ACM SIGCOMM Computer Communication Review, 2015. P. 183–197.
62. Amazon Redshift and PostgreSQL. Available at https://docs.aws.amazon.com/redshift/latest/dg/c_redshift-and-postgres-sql.html, accessed 07/17/2021.
63. Data warehouse system architecture. Available at https://docs.aws.amazon.com/redshift/latest/dg/c_high_level_system_architecture.html, accessed 07/17/2021.
64. *Anurag Gupta, Deepak Agarwal et al*. Amazon Redshift and the Case for Simpler Data Warehouses. Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, 2015. P. 1917–1923.
65. The Microsoft Modern Data Warehouse. White paper, 2016. Available at http://download.microsoft.com/download/C/2/D/C2D2D5FA-768A-49AD-8957-1A434C6C8126/Microsoft_Modern_Data_Warehouse_white_paper.pdf, accessed 07/18/2021.
66. Azure Synapse SQL architecture. Available at https://docs.microsoft.com/en-us/azure/synapse-analytics/sql/overview-architecture, accessed 07/18/2021.
67. What is Azure Synapse Analytics? Available at https://docs.microsoft.com/en-us/azure/synapse-analytics/overview-what-is, accessed 07/18/2021.
68. Use transactions in a SQL pool in Azure Synapse. Available at https://github.com/MicrosoftDocs/azure-docs/blob/master/articles/synapse-analytics/sql-data-warehouse/sql-data-warehouse-develop-transactions.md, accessed 07/18/2021.
69. Ashish Motivala, Jiaqi Yan. The Snowflake Elastic Data Warehouse, SIGMOD 2016 and beyond. Available at https://15721.courses.cs.cmu.edu/spring2018/slides/25-snowflake.pdf, accessed 07/18/2021.
70. *Benoit Dageville, Thierry Cruanes et al*. The Snowflake Elastic Data Warehouse. Proceedings of the 2016 International Conference on Management of Data, 2016. P. 215–226.
71. *Anastassia Ailamaki, David J. DeWitt et al*. Weaving Relations for Cache Performance. Proceedings of the 27th International Conference on Very Large Data Bases, September 2001. P. 169–180.
72. *David Karger, Eric Lehman et al*. Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. Proceedings of the twenty-ninth annual ACM symposium on Theory of computing, 1997. P. 654–663.
73. *Goetz Graefe*. The Cascades Framework for Query Optimization. IEEE Bulletin of the Technical Committee on Data Engineering. V. 18. № 3, 1995. P. 19–29.
74. *Franz Faerber, Alfons Kemper et al.* Main Memory Database Systems. Foundations and Trends in Databases. 2016. V. 8. № 1–2. P. 1–130.
75. *Frederik Transier, Peter Sanders*. Engineering basic algorithms of an in-memory text search engine. ACM Transactions on Information Systems, 2010, Article No. 2.
76. *J. Andrew Ross*. SAP NetWeaver BI Accelerator. SAP PRESS, 2008. 260 p.
77. *Sang K. Cha and Changbin Song*. P*TIME: Highly Scalable OLTP DBMS for Managing Update-Intensive Stream Workload. Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004. P. 1033–1044.
78. *André Bögelsack, Stephan Gradl, Manuel Mayer, Helmut Krcmar*. SAP MaxDB Administration. SAP PRESS, 2009. 326 p.
79. *Franz Faerber, Norman May et al.* The SAP HANA Database – An Architecture Overview. IEEE Bulletin of the Technical Committee on Data Engineering. 2012. V. 35. № 1. P. 28–33.
80. *Per-Åke Larson, Cipri Clinciu et al*. SQL Server Column Store Indexes. Proceedings of the ACM SIGMOD International Conference on Management of data, 2011. P. 1177–1184.
81. *Per-Åke Larson, Mike Zwilling, Kevin Farlee*. The Hekaton Memory-Optimized OLTP Engine. Bulletin of the Technical Committee on Data Engineering. 2013. V. 36. № 2. P. 34–40.
82. *Per-Åke Larson, Adrian Birka et al.* Real-Time Analytical Processing with SQL Server. Proceedings of the VLDB Endowment. 2015. V. 8. № 12. P. 1740–1751.
83. *Ahmed Eldawy, Justin Levandoski, Per-Åke Larson*. Trekking Through Siberia: Managing Cold Data in a Memory-Optimized Database. Proceedings of the VLDB Endowment. 2014. V. 7. № 11. P. 931–942.
84. *Tirthankar Lahiri, Marie-Anne Neimat, Steve Folkman*. Oracle TimesTen: An In-Memory Database for Enterprise Applications. Bulletin of the Technical Committee on Data Engineering. 2013. V. 36. № 2. P. 6–13.
85. *Sherry Listgarten and Marie-Anne Neimat*. Modelling Costs for a MM-DBMS. In Proceedings of the International Workshop on Real-Time Databases, Issues and Applications (RTDB), 1996. P. 72–78.
86. *Tirthankar Lahiri, Shasank Chavan et al*. Oracle Database In-Memory: A dual format in-memory database. 2015 IEEE 31st International Conference on Data Engineering, Seoul, 2015. P. 1253–1258.

87. *Niloy Mukherjee, Shasank Chavan et al.* Distributed Architecture of Oracle Database In-memory. Proceedings of the VLDB Endowment. 2015. V. 8. № 12. P. 1630–1641.
88. *Shasank Chavan, Gurmeet Goindi*. Oracle Database In-Memory on Exadata: A Potent Combination. Oracle OpenWorld 2018. Available at https://www.oracle.com/technetwork/database/exadata/pro4016-exadataandinmemory-5187037.pdf, accessed 07/18/2021.
89. *Ronald Barber, Peter Bendel et al.* Business Analytics in (a) Blink. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering. 2012. V. 35. № 1. P. 9–14.
90. IBM Informix Warehouse Accelerator. Technical white paper. URL: https://www.iiug.org/library/ids_12/IWA%20White%20Paper-2013-03-21.pdf, accessed 07/18/2021.
91. *Vijayshankar Raman, Gopi Attaluri et al.* DB2 with BLU Acceleration: So Much More than Just a Column Store. Proceedings of the VLDB Endowment. 2013. V. 6. № 11. P. 1080–1091.
92. *Whei-Jen Chen, Brigitte Bläser et al.* Architecting and Deploying DB2 with BLU Acceleration. IBM Redbooks, 2014. 420 p.
93. Faster analytics with Hyper. Available at https://www.tableau.com/products/new-features/hyper, accessed 07/18/2021.
94. *Alfons Kemper and Thomas Neumann*. HyPer – Hybrid OLTP&OLAP High Performance Database System. Technical Report, TUM-I1010, Munich Technical University, 2010. 29 p.
95. *Alfons Kemper, Thomas Neumann et al.* Transaction Processing in the Hybrid OLTP&OLAP Main-Memory Database System HyPer. Bulletin of the Technical Committee on Data Engineering. 2013. V. 36. № 2. P. 41–47.
96. *Martina-Cezara Albutiu, Alfons Kemper, Thomas Neumann*. Massively Parallel Sort-Merge Joins in Main Memory Multi-Core Database Systems. Proceedings of the VLDB Endowment. 2012. V. 5. № 10. P. 1064–1075.
97. *Thomas Neumann, Tobias Mühlbauer, Alfons Kemper*. Fast Serializable Multi-Version Concurrency Control for Main-Memory Database Systems. Proceedings of the ACM SIGMOD International Conference on Management of data, 2015. P. 677–689.
98. *Mihnea Andrei, Christian Lemke, et al.* SAP HANA Adoption of Non-Volatile Memory. Proceedings of the VLDB Endowment. 2017. V. 10. № 12. P. 1754–1765.
99. *Bob Dorr*. How It Works (It Just Runs Faster): Non-Volatile Memory SQL Server Tail of Log Caching on NVDIMM. Available at https://docs.microsoft.com/ru-ru/archive/blogs/bobsql/how-it-works-it-just-runs-faster-non-volatile-memory-sql-server-tail-of-log-caching-on-nvdimm, accessed 07/18/2021.
100. Oracle Database 20c. Database Administrator's Guide. Using Persistent Memory Database. Available at https://docs.oracle.com/en/database/oracle/oracle-database/20/admin/index.html, accessed 07/18/2021.
101. *Joy Arulraj, Andrew Pavlo*. Non-Volatile Memory Database Management Systems. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2019. 192 p.
102. *Ismail Oukid*. Architectural Principles for Database Systems on Storage-Class Memory. Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn, 2019. P. 477–486.