Содержание

Об авторах

М. Мамер Ёсу – профессор Черитонской школы компьютерных наук в университете Ватерлоо в Канаде. Исследованиями в области распределенного управления данными он занимается уже тридцать лет. Состоит членом Королевского общества Канады, Американской ассоциации содействия развитию науки (AAAS), Ассоциации по вычислительной технике (ACM) и Института инженеров по электротехнике и электронике (IEEE). Является избранным членом Турецкой академии наук и членом общества «Сигма Кси». Является лауреатом премии за прижизненные достижения Канадского общества компьютерных наук за 2019 год, премии за проверенные временем достижения ACM SIGMOD за 2015 год, премии за вклад в науку ACM SIGMOD за 2008 год и премии выдающимся выпускникам технического колледжа университета штата Огайо за 2008 год. Также получил две премии за лучшую работу и один похвальный отзыв на публикацию. Состоит в редколлегиях многих журналов и книжных серий, наряду с Линь Лю является одним из главных редакторов «Энциклопедии по системам баз данных».

Патрик Вальдуриес – главный научный сотрудник французской компании Inria. Преподавал информатику в университете Пьера и Мари Кюри (UPMC) в Париже (2000–2002) и занимал должность исследователя в компании Microelectronics and Computer Technology Corp. в Остине, штат Техас (1985–1989). Начиная с 2019 года является научным консультантом стартапа LeanXcale.

В настоящее время возглавляет команду Zenith (включающую сотрудников компании Inria и университета Монпелье, LIRMM), которая занимается наукой о данных, в частности управлением данными в крупномасштабных распределенных и параллельных системах и управлением научными данными. Является заместителем редактора в нескольких журналах, включая *VLDB Journal*, *Distributed and Parallel Databases* и *Internet and Databases*.

Занимал место в правлении таких крупных конференций, как SIGMOD и VLDB. Исполнял обязанности председателя конференций SIGMOD 2004, EDBT 2008 и VLDB 2009. Получил несколько наград за лучшую работу, в т. ч. на VLDB 2000. Лауреат премии по информатике от французского подразделения IBM за 1993 год и премии компании Inria, Французской академии наук и компании Dassault Systems за инновации в 2014 году. Является действительным членом ACM.

Предисловие

Первое издание этой книги вышло в 1991 году, когда технология была новой, а продуктов не так много. В предисловии к первому изданию мы цитировали Майкла Стоунбрейкера, который в 1988 году говорил, что в следующие 10 лет централизованные СУБД станут «антикварной редкостью», а большая часть организаций перейдет на распределенные СУБД. Это предсказание, конечно же, сбылось, в современном мире значительная доля систем либо распределенные, либо параллельные – обычно для их описания употребляется термин «горизонтально масштабируемые». Когда мы собирали материал для первого издания, курсы по базам данных для студентов и магистрантов были не так распространены, как сейчас, поэтому книга содержала пространные сведения о централизованных решениях, предварявшие обсуждение распределенных и параллельных систем. Но и на этом фронте произошли большие перемены, теперь трудно встретить магистранта, не имеющего хотя бы начальных знаний о технологии баз данных. Поэтому учебник по технологии распределенных и параллельных баз данных для магистрантов сейчас нужно позиционировать иначе. Такую цель мы и поставили себе в этом издании, сохранив, впрочем, многие новые темы, появившиеся в третьем издании. Перечислим основные изменения, внесенные в четвертое издание.

1. С годами побудительные мотивы этой технологии и среда ее развертывания претерпели изменения (веб, облако и т. д.). Поэтому вводная глава нуждалась в серьезном пересмотре. Мы переписали введение, отразив современный взгляд на технологию.
2. Мы добавили новую главу, посвященную обработке больших данных, в которую включили распределенные системы хранения, потоковую обработку данных, платформы MapReduce и Spark, анализ графов и озера данных. По мере распространения таких систем приобретает важность систематическое изложение этих вопросов.
3. Аналогично мы учли растущее влияние NoSQL-систем, посвятив им отдельную главу. В ней дается обзор четырех типов баз данных NoSQL (хранилища ключей и значений, документные хранилища, столбцовые базы данных и графовые СУБД), а также NewSQL-систем и полихранилищ.
4. Мы объединили главы об интеграции баз данных и обработке запросов к нескольким базам в одну главу.
5. Мы кардинально переработали изложение вопроса об обработке данных в вебе, сместив акцент с XML на технологию RDF, которая сейчас больше распространена. В эту главу мы включили обсуждение подходов к интеграции веб-данных, в т. ч. важный вопрос о качестве данных.
6. Мы также подвергли ревизии главу об одноранговой обработке данных и включили подробное объяснение технологии блокчейна.
7. Стремясь вычистить предыдущие главы, мы ужали главы, относящиеся к обработке запросов и управлению транзакциями, исключив принципиально централизованные методы и сосредоточив внимание на рас-

пределенных и параллельных. Попутно мы включили некоторые темы, которые приобрели большое значение, в частности динамическую обработку запросов (вихревых операторов), а также алгоритм консенсуса Paxos и его применение в протоколах фиксации.

8. Мы исправили главу о параллельных СУБД, уточнив цели, в частности пояснили различие между горизонтальным и вертикальным масштабированием и обсудили параллельные архитектуры, в т. ч. UMA и NUMA. Также добавлен новый раздел о параллельных алгоритмах сортировки и вариантах параллельных алгоритмов соединения, в которых задействованы преимущества памяти большого объема и многоядерных процессоров, которые ныне распространены повсеместно.
9. Мы пересмотрели главу о проектировании распределенности, включив пространное обсуждение современных подходов, сочетающих фрагментацию и размещение. После реорганизации материала эта глава стала центральным источником информации по секционированию данных во всех обсуждениях распределенного и параллельного управления данными.
10. Хотя объектные технологии по-прежнему играют роль в информационных системах, их значимость в системах распределенного и параллельного управления данными снизилась. Поэтому в этом издании мы исключили главу об объектных базах данных.

Очевидно, что после переработки книга в целом и каждая глава в отдельности стали ближе к современному состоянию дел. Исключенный материал не утрачен вовсе, а оставлен в виде онлайновых приложений на веб-странице книги по адресу https://cs.uwaterloo.ca/ddbs. Мы решили перевести его в сеть и не включать в печатную версию, чтобы сохранить разумный размер (и цену) книги. На сайте имеются также слайды, которыми можно пользоваться в учебном курсе по материалам книги, и решения большинства упражнений (доступны только преподавателям, решившим читать курс на основе этой книги).

Как и в случае предыдущих изданий, в редактировании книги нам помогали многие коллеги, которым мы выражаем благодарность (не придерживаясь при перечислении какого-то определенного порядка). Дэн Олтеану (Dan Olteanu) предложил в главе 3 изящное обсуждение двух оптимизаций, которые могут значительно уменьшить время обслуживания материализованных представлений. Фил Бернштейн (Phil Bernstein) предоставил предварительные материалы к новым статьям о многоверсионном управлении транзакциями, легшие в основу обновленного обсуждения этой темы в главе 5. Хузаима Дауджи (Khuzaima Daudjee) также помог в подготовке списка более современных публикаций по распределенной обработке транзакций, который мы включили в библиографические ссылки к той же главе. Рикардо Хименес-Перис (Ricardo Jimenez-Peris) предоставил материал по высокопроизводительным транзакционным системам, включенный туда же. Деннис Шаша (Dennis Shasha) отредактировал новый раздел по блокчейну в главе по одноранговым системам. Майкл Кэри (Michael Carey) прочитал главы по большим данным, NoSQL, NewSQL, полихранилищам и параллельным СУБД и дал весьма подробные замечания, благодаря которым эти главы стали значительно лучше.

Студенты Тамера, Анли Пачачи (Anil Pacaci), Халед Аммар (Khaled Ammar) и аспирант Сяофей Чжан (Xiaofei Zhang) написали подробные рецензии на главу по большим данным, и части их текстов включены в эту главу. В главу по NoSQL, NewSQL и полихранилищам включены части публикаций Бояна Колева (Boyan Kolev) и студентки Патрика Карлины Бондиомбой (Carlyna Bondiombouy). Джим Веббер (Jim Webber) отредактировал раздел по Neo4j этой главы. Характеристика графовых аналитических систем, приведенная в этой главе, частично основана на магистерской диссертации Миньянь Хана, в которой он заодно предложил подход на основе GiraphUC, обсуждаемый там же. Части этой главы прочитали и оставили весьма полезные замечания также Семих Салихоглу (Semih Salihoglu) и Лукаш Голаб (Lukasz Golab). Алон Халеви поделился комментариями по поводу обсуждения веб-таблиц в главе 12. Обсуждение качества данных в процессе интеграции веб-данных написали Ихаб Ильяс (Ihab Ilyas) и Су Чу (Xu Chu). Стратос Идреос (Stratos Idreos) пояснил, как можно использовать крекинг базы данных в качестве подхода к секционированию, его текст включен в главу 2. Ренан Соуза (Renan Souza) и Фабиан Штётер (Fabian Stöter) просмотрели всю книгу целиком.

В третье издание книги было включено много новых тем, которые перекочевали в это издание, при написании этих глав большой вклад внесли многие наши коллеги. Мы хотим еще раз отметить их содействие, поскольку его влияние прослеживается и в новом издании. Рене Миллер (Renée Miller), Эрхард Рам (Erhard Rahm) и Алон Халеви (Alon Halevy) многое сделали для сведения воедино материалов по интеграции баз данных, которые затем были внимательно отрецензированы Авигдором Галом (Avigdor Gal). Матиас Ярке (Matthias Jarke), Сянь Ли (Xiang Li), Готфрид Фоссен (Gottfried Vossen), Эрхард Рам и Андреас Тор (Andreas Thor) предложили упражнения к этой главе. Хуберт Нааке (Hubert Naacke) внес вклад в раздел о моделировании гетерогенной стоимости, а Фабио Порто (Fabio Porto) – в раздел об адаптивной обработке запросов. Главу 6 о репликации данных невозможно было бы написать без помощи Густаво Алонсо (Gustavo Alonso) и Беттины Кемме (Bettina Kemme). Эстер Пачитти (Esther Pacitti) также внесла вклад в главу о репликации данных – прочитав ее и предложив вводные материалы; она же помогала в написании раздела о кластерах баз данных в главе о параллельных СУБД. При работе над главой об одноранговой обработке данных мы много беседовали с Бень Чин Ои (Beng Chin Ooi). В разделе этой главы, посвященном обработке запросов в одноранговых системах, использованы материалы из докторской диссертации Резы Акбаринья (Reza Akbarinia) и Венцеслао Пальма (Wenceslao Palma), а в разделе о репликации – материалы из докторской диссертации Видала Мартинса (Vidal Martins).

Мы благодарны нашему редактору в издательстве Springer Сюзан Лагерстром-Файф (Susan Lagerstrom-Fife) за продвижение проекта в самом издательстве и за постоянные напоминания о необходимости закончить работу вовремя. Мы пропустили почти все назначенные ей крайние сроки, но надеемся, что результат получился неплохим.

Наконец, нам было бы очень интересно услышать ваши замечания и предложения. Мы приветствуем любые отзывы, но особенно нас интересует ваше мнение по следующим вопросам:

1) любые ошибки и опечатки, просочившиеся, несмотря на все наши усилия (хочется надеяться, что их немного);
2) темы, которые следовало бы исключить, и, наоборот, темы, которые нужно добавить или изложить более подробно;
3) придуманные вами упражнения, которые вы хотели бы включить в книгу.

Ватерлоо, Канада — М. Тамер Ёсу (tamer.ozsu@uwaterloo.ca)
Монпелье, Франция — Патрик Вальдуриес (patrick.valduriez@inria.fr)
июнь 2019

От издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв на нашем сайте www.dmkpress.com, зайдя на страницу книги и оставив комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу dmkpress@gmail.com; при этом укажите название книги в теме письма.

Если вы являетесь экспертом в какой-либо области и заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу http://dmkpress.com/authors/publish_book/ или напишите в издательство по адресу dmkpress@gmail.com.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте www.dmkpress.com на странице с описанием соответствующей книги.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы обеспечить высокое качество наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг, мы будем очень благодарны, если вы сообщите о ней главному редактору по адресу dmkpress@gmail.com. Сделав это, вы избавите других читателей от недопонимания и поможете нам улучшить последующие издания этой книги.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Springer очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконной публикацией какой-либо из наших книг, пожалуйста, пришлите нам ссылку на интернет-ресурс, чтобы мы могли применить санкции.

Ссылку на подозрительные материалы можно прислать по адресу электронной почты dmkpress@gmail.com.

Мы высоко ценим любую помощь по защите наших авторов, благодаря которой мы можем предоставлять вам качественные материалы.

Глава 1

Введение

Современная вычислительная среда в значительной степени распределенная – компьютеры подключены к интернету и образуют всемирную распределенную систему. В организациях имеются территориально распределенные связанные между собой центры обработки данных (ЦОД), насчитывающие сотни, а то и тысячи компьютеров, объединенных в высокоскоростную сеть, которая содержит как распределенные, так и параллельные системы (рис. 1.1). Объем данных, хранимых в такой среде, возрос многократно. Не все данные хранятся в системах баз данных (на самом деле там хранится лишь малая их часть), но возникает желание так или иначе управлять этим распределенным по большой территории массивом данных. Это и есть задача распределенных и параллельных систем баз данных, которые несколько десятилетий назад занимали лишь небольшую нишу в мировой вычислительной среде, а теперь вышли на авансцену. В данной главе мы дадим общий обзор этой технологии, а в последующих займемся деталями.

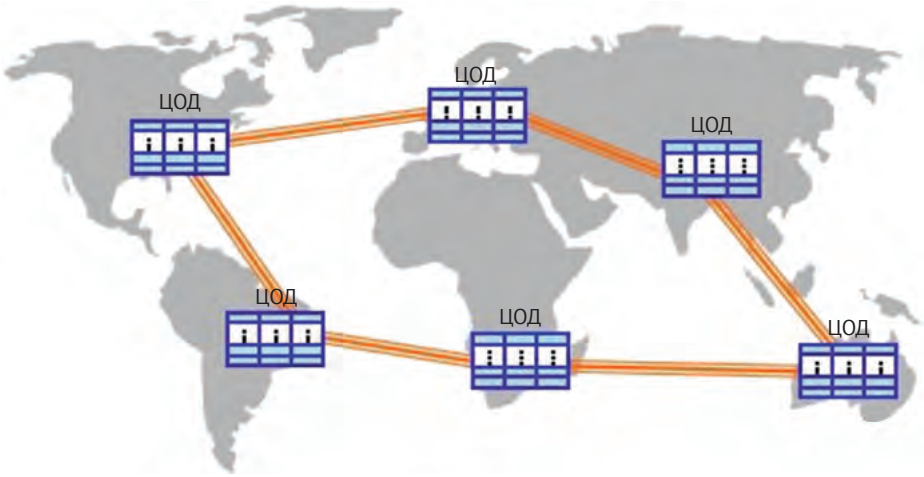

Рис. 1.1 ❖ Территориально распределенные центры обработки данных

1.1. Что такое распределенная система баз данных?

Распределенную базу данных мы определяем как *набор из нескольких логически взаимосвязанных баз данных, расположенных в узлах распределенной системы*. *Распределенной системой управления базами данных* (распределенной СУБД) мы далее называем *программную систему, которая допускает управление распределенной базой данных и делает распределенность прозрачной для пользователей*. Иногда, говоря «распределенная система баз данных» (распределенная СУБД), имеют в виду как распределенную базу данных, так и распределенную СУБД в нашем понимании. Мы выделяем две важные характеристики: логическая взаимосвязанность данных и их нахождение в распределенной системе.

Существование распределенной системы – важный момент. В этом контексте под *распределенной вычислительной системой* понимается несколько связанных между собой автономных обрабатывающих элементов (ОЭ). Возможности обрабатывающих элементов могут различаться, они могут быть гетерогенными, связи между ними тоже могут быть различными, но важно то, что ОЭ не имеют прямого доступа к состоянию друг друга, а могут узнать его, лишь обмениваясь сообщениями и, следовательно, неся затраты на коммуникацию. Поэтому если данные распределены, то доступ к ним и управление ими логически непротиворечивым способом требует особого внимания со стороны распределенной СУБД.

Распределенная СУБД – это не просто «набор файлов», которые можно по отдельности хранить в каждом ОЭ распределенной системы (обычно называемом «узлом» распределенной СУБД); данные в распределенной СУБД взаимосвязаны. Мы не будем конкретизировать, что означает «взаимосвязаны», поскольку требования зависят от типа данных. Например, в случае реляционных данных различные отношения или их части могут храниться в разных узлах (подробнее об этом см. главу 2), и для ответа на запросы, выражаемые, как правило, на языке SQL, требуется выполнять операции соединения или объединения. Обычно можно определить *схему* таких распределенных данных. На другом полюсе находятся данные в системах NoSQL (см. главу 11), в которых возможно гораздо менее ограничительное определение взаимосвязанности; например, это могут быть вершины графа, хранящиеся в разных узлах.

Подводя итоги этому обсуждению, можно сказать, что распределенная СУБД *логически едина*, но *физически распределена*. Это означает, что пользователь, работающий с распределенной СУБД, воспринимает ее как единую базу данных, хотя составляющие ее данные физически находятся в разных местах.

Как уже было сказано, обычно рассматриваются два типа распределенных СУБД: территориально распределенные (или *георапределенные*) и сосредоточенные в одном месте (одноузловые). В первом случае узлы соединены между собой глобальной сетью, для которой характерны большие задержки при передаче сообщений и более высокая частота ошибок. А во втором речь идет о системах, в которых ОЭ находятся близко друг к другу, так что

обмен сообщениями происходит гораздо быстрее (современные технологии позволяют считать задержку пренебрежимо малой) и с очень низкой частотой ошибок. Одноузловые распределенные СУБД обычно размещаются на кластерах компьютеров в одном ЦОДе и называются параллельными СУБД (их ОЭ по-английски называются «node» – в отличие от «site», а по-русски в обоих случаях употребляется термин «узел»). Выше отмечалось, что сегодня легко встретить распределенные СУБД, состоящие из нескольких одноузловых кластеров, соединенных глобальной сетью, т. е. имеет место гибридная многоузловая система. В этой книге мы в основном будем рассматривать задачи управления данными в узлах геораспределенной СУБД, а о проблемах одноузловых систем речь пойдет только в главах 8, 10 и 11, где обсуждаются параллельные СУБД, большие данные и системы NoSQL/NewSQL.

1.2. История распределенных СУБД

До появления систем баз данных в 1960-х годах каждое приложение само определяло свои данные и управляло ими (рис. 1.2). В этом режиме приложение принимало решения о структуре и методах доступа к данным и отвечало за управление файлом в системе хранения. Это приводило к значительной и неконтролируемой избыточности данных и высоким трудозатратам программистов, вынужденных заниматься управлением данными в приложениях.

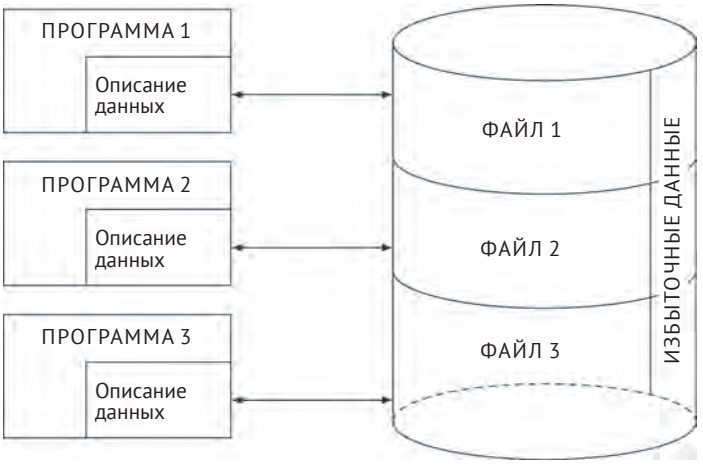

Рис. 1.2 ❖ Традиционная обработка файлов

Система баз данных позволяет определять и администрировать данные централизованно (рис. 1.3). Этот новый подход ведет к *независимости данных*, когда прикладная программа безразлична к изменению логической или физической организации данных, и наоборот. Таким образом, программисты освобождаются от ответственности за управление и сопровождение нужных им данных, а избыточность можно устранить (или уменьшить).

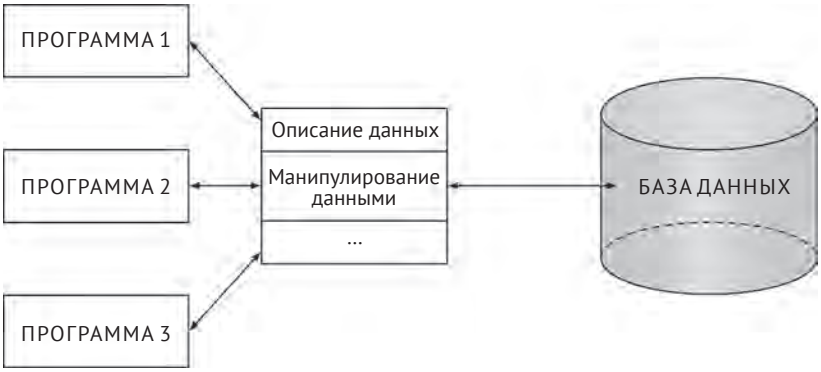

Рис. 1.3 ❖ Обработка базы даннх

Одним из побудительных мотивов использования систем баз данных являлось желание объединить всю информацию о работе предприятия и предоставить интегрированный и контролируемый доступ к этим данным. Мы сознательно употребляем термин «интегрированный», а не «централизованный», потому что данные, как уже было сказано, могут размещаться на разных территориально разнесенных компьютерах. Именно в этом смысл технологии распределенных баз данных. Мы уже отмечали, что физически распределенная система может находиться в одном месте или в нескольких местах. Поэтому каждый узел на рис. 1.5 может представлять собой центр обработки данных, соединенных сетью с другими центрами. Именно распределенные среды такого типа являются типичными в настоящее время, и именно их мы будем изучать в этой книге.

С годами архитектура распределенной системы баз данных претерпела значительные изменения. Такие ранние распределенные системы баз данных, как Distributed INGRES и SDD-1, проектировались как территориально распределенные системы с очень медленными сетевыми соединениями, поэтому они всеми силами стремились оптимизировать операции, чтобы уменьшить обмен данными по сети. Это были первые *одноранговые системы* (peer-to-peer – P2P) в том смысле, что все узлы имели похожую функциональность в части управления данными. С развитием персональных компьютеров и рабочих станций стала преобладать модель распределения типа *клиент-сервер*, в которой данные переместились на тыловой сервер, а пользовательские приложения работали на фронтальных рабочих станциях. Особенно часто такие системы стали развертываться на одной территории, где можно было обеспечить более высокую скорость сети и, стало быть, более частое взаимодействие между клиентами и сервером (или серверами). В 2000-х годах произошло возрождение P2P-систем, в которых исчезло различие между клиентскими и серверными компьютерами. Современные P2P-системы отличались от ранних в нескольких важных отношениях, которые мы обсудим ниже в этой главе. Все эти архитектуры существуют и сегодня и обсуждаются в последующих главах.

Становление Всемирной паутины (или просто веба) как основной платформы для совместной работы и обмена файлами оказало громадное влияние

на исследования по управлению распределенными данными. Стало доступно гораздо больше данных, но это не были тщательно структурированные и точно определенные данные, как в типичной СУБД; они были слабо структурированы или не структурированы вовсе (т. е. какую-то структуру они имели, но определенную не на уровне схемы базы данных), их происхождение было неизвестно (т. е. данные могли быть «грязными» или ненадежными), и зачастую они были противоречивы. Ко всему прочему многие данные хранились в системах, к которым не было простого доступа (в *скрытом вебе*). Поэтому усилия по распределенному управлению данными направляются на доступ к этим данным осмысленными способами.

Это направление развития стимулировало исследования в области *интеграции баз данных* – дисциплине, которая существовала с самого начала работ по распределенным базам данных. Первоначально эти усилия были направлены на поиск способов доступа к данным в различных базах (отсюда и термины *федеративная база данных* и *мультибаза данных*), но с появлением веб-данных фокус сместился в сторону виртуальной интеграции данных разных типов (поэтому термин *интеграция данных* стал более популярным). Сейчас в моде термин *озеро данных*, который подразумевает, что все данные собираются в логически едином хранилище, из которого каждое приложение извлекает нужные ему данные. Мы обсудим интеграцию данных в главе 7, а озера данных – в главах 10 и 12.

За последние десять лет важным явлением стали облачные вычисления. Под этим понимается вычислительная модель, в которой ряд поставщиков услуг предоставляют в общее пользование разделяемые и территориально распределенные вычислительные ресурсы, так что пользователи могут арендовать их по мере необходимости. Клиенты могут взять в аренду базовую вычислительную инфраструктуру для разработки собственного программного обеспечения, а затем решить, какую операционную систему предпочитают, и создать для себя виртуальные машины (ВМ) со средой, в которой хотят работать, – такой подход называется *инфраструктура как услуга* (IaaS). Возможна и более развитая облачная среда, в которой в аренду сдается не только базовая инфраструктура, но и вся вычислительная платформа, на которой клиенты разрабатывают свое ПО, – это *платформа как услуга* (PaaS). Самый продвинутый вариант – когда поставщик услуг предоставляет в аренду конкретное программное обеспечение, этот подход называется *программное обеспечение как услуга* (SaaS). В последнее время начинают предлагать услуги управления распределенной базой данных в облаке как часть SaaS.

Мы дадим общий обзор всех этих архитектур в разделе 1.6.1.2, а затем обсудим их в отдельных главах более подробно.

1.3. Различные способы доставки данных

В распределенных базах данных доставка данных производится между узлами – либо от серверных узлов клиентским в ответ на запросы, либо между несколькими серверными узлами. Мы будем характеризовать варианты

доставки данных по трем независимым осям: *режимы доставки*, *частота* и *методы коммуникации*. Комбинирование всех трех свойств дает проектировщику достаточно богатый выбор.

Существует три режима доставки: вытягивание, проталкивание и гибридный. В режиме вытягивания передача данных инициируется узлом в виде запроса поставщику данных – это может быть клиент, запрашивающий данные у сервера, или сервер, запрашивающий данные у другого сервера. Далее мы будем употреблять термины «получатель» и «поставщик» для обозначения компьютеров, которые получают и отправляют данные соответственно. Получив запрос, поставщик находит и передает данные. Основная характеристика доставки методом вытягивания заключается в том, что получатель узнаёт о наличии новых или обновленных данных у поставщика только после того, как явно спросит об этом. Кроме того, в режиме вытягивания работа поставщика постоянно прерывается новыми запросами. У этого режима есть ограничение – получатель имеет только те данные, о которых знает, и только тогда, когда попросит. Традиционные СУБД предлагают в основном доставку данных в режиме вытягивания.

В режиме проталкивания передачу данных инициирует поставщик без явного запроса. Основная трудность такого подхода – решить, какие данные будут представлять всеобщий интерес и когда отправлять их потенциально заинтересованным получателям: периодически, нерегулярно или по некоторому условию. Таким образом, полезность проталкивания сильно зависит от того, насколько точно поставщик умеет предсказывать нужды получателей. В режиме проталкивания поставщики распространяют информацию либо неограниченному кругу получателей (ненаправленное вещание), которые прослушивают передающую среду, либо избранному множеству получателей (групповое вещание), принадлежащему определенным категориям.

Гибридный режим сочетает механизмы вытягивания и проталкивания. Один из способов комбинирования того и другого – постоянный запрос (см. раздел 10.3), когда сначала в режиме вытягивания (посредством отправки запроса) инициируется первая передача данных от поставщика клиента, а затем поставщик уже сам отправляет обновленные данные в режиме проталкивания.

Существует три типичные частоты доставки данных, характеризующие ее регулярность: периодическая, условная и ситуативная (или нерегулярная).

При *периодической* доставке данные отправляются поставщиками с регулярными интервалами. Величина интервала может задаваться по умолчанию системой или в профиле каждого получателя. Периодическая доставка возможна как в режиме вытягивания, так и в режиме проталкивания. Она выполняется по заранее заданному регулярному расписанию. Примером периодического вытягивания является еженедельный запрос цены акций компании, а периодического проталкивания – отправка цен акций приложением по расписанию, скажем каждое утро. Периодическое проталкивание особенно полезно в ситуациях, когда получатель доступен не постоянно или не всегда может отреагировать на присланные данные, как, скажем, в случае, когда клиент установлен в мобильном устройстве, которое иногда отключается от сети.

В случае *условной доставки* данные отправляются поставщиком, когда выполнены некоторые условия, заданные в профиле получателей. Это может быть простое условие типа истечения промежутка времени или сложные правила вида событие–условие–действие. Условная доставка чаще всего используется в системах, работающих в гибридном режиме или в режиме вытягивания. В этом случае данные рассылаются при выполнении предопределенного условия, а не по расписанию. Примером условного проталкивания может служить приложение, которое отправляет цены акций, только когда они изменяются. Примером гибридной условной доставки является приложение, которое отправляет выписку об остатке на счете, только когда остаток оказывается на 5 % меньше оговоренного порога. Условное проталкивание предполагает, что изменения критически важны для получателей, которые постоянно ожидают их и должны отреагировать на полученные данные. Гибридное условное проталкивание дополнительно предполагает, что получатели могут смириться с пропуском какой-то информации об обновлениях.

Ситуативная доставка выполняется нерегулярно и в основном в системах, работающих в режиме вытягивания. Данные вытягиваются у поставщиков по ситуации в ответ на запросы. Напротив, периодическое вытягивание имеет место, когда запрашивающая сторона производит опрос поставщиков по расписанию.

Третья характеристика доставки информации – метод коммуникации. Он определяет, каким образом поставщики и получатели взаимодействуют с целью доставки информации клиентам. Вариантов два: одноадресная передача и передача один-ко-многим. В *одноадресном* режиме поставщик посылает данные одному получателю в определенном режиме доставки и с определенной частотой. При передаче один-ко-многим поставщик посылает данные сразу нескольким получателям. Отметим, что речь не идет о конкретном протоколе: в режиме один-ко-многим может применяться как протокол группового вещания, так и протокол широковещания.

Следует отметить, что описанная классификация – предмет споров. Неочевидно, что все точки пространства вариантов имеют смысл. Кроме того, придание смысла некоторым комбинациям, например условная **и** периодическая (быть может, и осмысленная), наталкивается на трудности. Однако в первом приближении она может охарактеризовать сложность систем управления распределенными данными. В этой книге нас будут интересовать в первую очередь системы ситуативной доставки данных в режиме вытягивания, но мы обсудим режим проталкивания и гибридный режим в разделе 10.3, посвященном потоковым системам.

1.4. Обещания распределенных СУБД

Можно привести много преимуществ распределенных СУБД, но в основном они сводятся к четырем фундаментальным положениям, которые можно рассматривать как обещания технологии: прозрачное управление распределенными и реплицированными данными, надежный доступ к данным по-

средством распределенных транзакций, улучшенная производительность и простое расширение системы. В этом разделе мы обсудим эти четыре обещания и попутно введем ряд понятий, который будем изучать в последующих главах.

1.4.1. Прозрачное управление распределенными и реплицированными данными

Под прозрачностью понимается отделение высокоуровневой семантики системы от низкоуровневых вопросов реализации. Иными словами, прозрачная система «скрывает» детали реализации от пользователей. Преимущество полностью прозрачной СУБД – высокий уровень поддержки, оказываемой разработке сложных приложений. Прозрачность в распределенных СУБД можно рассматривать как обобщение концепции независимости данных в централизованных СУБД (об этом еще будет сказано ниже).

Начнем обсуждение с примера. Рассмотрим конструкторскую компанию, имеющую отделения в Бостоне, Ватерлоо, Париже и Сан-Франциско. В каждом отделении выполняются какие-то проекты, и компания хочет иметь базу данных работников, проектов и связанных с этим данных. В предположении, что база данных реляционная, мы можем хранить эту информацию в нескольких отношениях (рис. 1.4): в EMP хранятся данные о работнике – номер, имя и должность[1], в PROJ – данные о проектах (атрибут LOC содержит отделение, в котором выполняется проект). В PAY хранятся сведения о зарплатах (в предположении, что два человека, занимающих одинаковые должности, получают одинаковую зарплату), а в ASG – сведения о распределении людей по проектам (DUR – продолжительность назначения, RESP – функция данного человека в данном проекте). Если бы все эти данные хранились в централизованной СУБД и мы хотели бы найти имена и зарплаты тех, кто работает над одним проектом больше 12 месяцев, то написали бы такой SQL-запрос:

```
SELECT ENAME, SAL
FROM EMP NATURAL JOIN ASG, EMP NATURAL JOIN PAY
WHERE ASG.DUR > 12
```

EMP(ENO, ENAME, TITLE)
PROJ(PNO, PNAME, BUDGET, LOC)
ASG(ENO, PNO, RESP, DUR)
PAY(TITLE, SAL)

Рис. 1.4 ❖ Пример базы данных конструкторской компании

Однако с учетом распределенной природы бизнеса этой компании предпочтительно локализовать данные, так чтобы сведения о работающих в офисе Ватерлоо хранились в Ватерлоо, сведения о работающих в бостонском офи-

[1] Первичные ключи подчеркнуты.

се – в Бостоне и т. д. То же самое относится к данным о проектах и зарплатах. Таким образом, дело свелось к процессу разбиения каждого отношения на части и хранению разных частей в разных узлах. Этот процесс называется *секционированием*, или *фрагментацией данных*, мы подробно рассмотрим его в главе 2.

Далее, иногда предпочтительно дублировать часть данных в других узлах ради повышения производительности и надежности. В результате мы получаем фрагментированную и реплицированную распределенную базу данных (рис. 1.5). Полностью прозрачный доступ означает, что пользователь может сформулировать запрос именно так, как показано выше, не обращая внимания на фрагментацию, местоположение или репликацию данных, и оставить решение этих вопросов системе. Чтобы система могла адекватно обработать запрос такого типа к распределенной, фрагментированной и реплицированной базе данных, она должна реализовывать различные типы прозрачности, о которых мы и поговорим ниже.

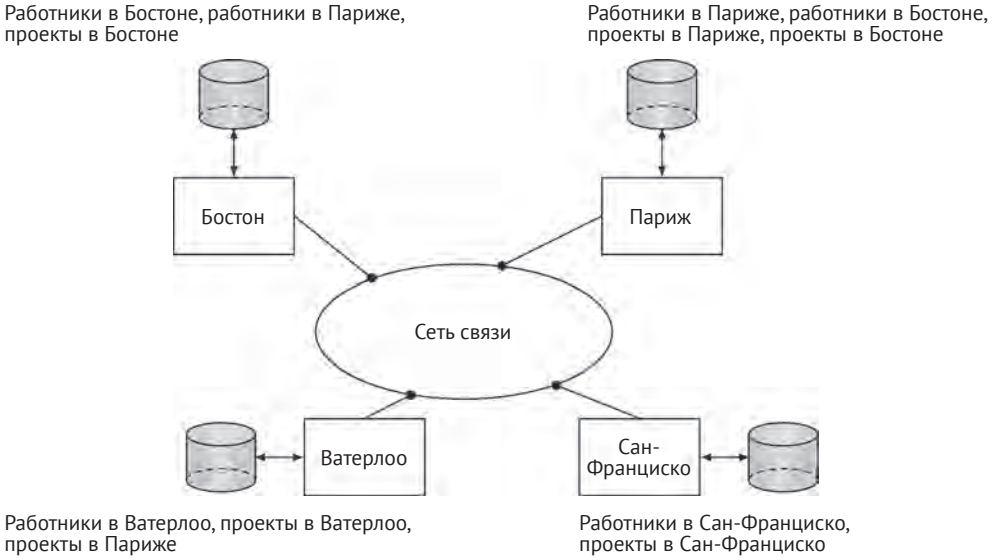

Рис. 1.5 ❖ Распределенная база данных

Независимость данных. Эта концепция, заимствованная у централизованных СУБД, относится к независимости пользовательских приложений от изменений в определении и организации данных, и наоборот.

Обычно говорят о двух типах независимости данных: логической и физической. Под *логической независимостью данных* понимается независимость приложений от изменений в логической структуре (т. е. схеме) базы данных. С другой стороны, *физическая независимость данных* связана с сокрытием деталей структуры хранения от пользовательских приложений. После того как приложение написано, его не должны волновать детали физической организации данных. Поэтому не должно возникать необходимости в модифи-

кации приложения, когда организация данных изменяется из соображений производительности.

Прозрачность сети. Желательно, чтобы пользователи были экранированы от деталей работы сети связи, соединяющей узлы, хорошо бы даже, чтобы само существование сети было скрыто. Тогда не будет разницы между приложениями, работающими с централизованной и распределенной базами данных. Такой вид прозрачности называется *прозрачностью сети*, или *прозрачностью распределения*.

Иногда выделяют два типа прозрачности распределения: прозрачность местоположения и прозрачность именования. Под *прозрачностью местоположения* понимается тот факт, что команда, необходимая для выполнения задачи, не зависит ни от местоположения данных, ни от системы, в котором операция выполняется. *Прозрачность именования* означает, что каждому объекту в базе данных присвоен уникальный идентификатор. В отсутствие прозрачности именования пользователи должны включать имя (или идентификатор) местоположения в состав имени объекта.

Прозрачность фрагментации. Как уже было сказано, часто желательно разбить каждое отношение базы данных на меньшие фрагменты и обращаться с каждым фрагментом как с отдельным объектом базы данных (т. е. отдельным отношением). Обычно это делают из соображений производительности, доступности и надежности – более подробное обсуждение см. в главе 2. Было бы хорошо, чтобы пользователи не знали о фрагментации при формулировании запросов, а система умела отображать запрос, сформулированный в терминах полных отношений, описанных в схеме, на множество запросов, выполняемых над частичными отношениями. Иными словами, требуется найти стратегию обработки запросов, основанную на фрагментах, а не на отношениях, пусть даже запросы формулируются в терминах последних.

Прозрачность репликации. Из соображений производительности, надежности и доступности обычно желательно иметь возможность распределять данные посредством репликации между машинами в сети. В предположении, что данные реплицируются, возникает вопрос, должны ли пользователи знать о существовании копий или система будет обрабатывать их самостоятельно, а пользователь должен действовать так, будто имеется всего одна копия данных (заметим, что мы говорим не о местоположении копий, а только об их существовании). С точки зрения пользователя, предпочтительно не думать о копиях и никак не упоминать тот факт, что некоторое действие может или должно применяться к нескольким копиям. Вопрос о репликации в распределенной базе данных поднимается в главе 2 и подробно обсуждается в главе 6.

1.4.2. Обеспечение надежности с помощью распределенных транзакций

Распределенные СУБД призваны повысить надежность, поскольку в них имеются реплицированные части, а значит, исключается единая точка отказа. Отказа одного узла или отказа линии связи, из-за которого один или не-

сколько узлов становятся недоступными, недостаточно для выхода из строя системы в целом. В случае распределенной базы данных это означает, что некоторые данные могут оказаться недоступными, но при должной аккуратности пользователям будет разрешено обращаться к другим частям распределенной базы данных. Под «должной аккуратностью» обычно понимается поддержка распределенных транзакций.

СУБД с полной поддержкой транзакций гарантирует, что одновременное выполнение транзакций несколькими пользователями не приведет к несогласованности базы данных, т. е. каждый пользователь может считать, что только его запрос выполняется базой (*прозрачность конкурентности*) даже при наличии отказов системы (*прозрачность отказов*), при условии что каждая транзакция корректна, т. е. не нарушает правил целостности, заданных в базе данных.

Для поддержки транзакций необходимо реализовать управление конкурентностью транзакций и распределенные протоколы обеспечения надежности, в частности протоколы двухфазной фиксации (2PC) и распределенного восстановления. Эти протоколы значительно сложнее, чем в централизованных СУБД, они обсуждаются в главе 5. Для поддержки реплик необходима реализация протоколов управления репликами, которые следят за соблюдением семантики доступа к ним. Эти протоколы обсуждаются в главе 6.

1.4.3. Повышенная производительность

Повышение производительности распределенной СУБД обычно обусловлено двумя моментами. Во-первых, распределенная СУБД фрагментирует базу данных, давая возможность хранить данные поблизости от точек их использования (*локальность данных*). Такой подход дает два преимущества:

1) поскольку каждый узел обрабатывает только часть базы данных, конкуренция за процессор и службы ввода-вывода не так сильна, как в централизованных базах;
2) благодаря локальности уменьшаются задержки удаленного доступа, обычно свойственные глобальным сетям.

Этот момент связан с накладными расходами на распределенные вычисления, когда данные находятся в удаленных узлах, к которым приходится обращаться по сети. Считается, что при таких условиях лучше приблизить функциональность управления данными к месту их расположения, чем передавать большие объемы данных. Иногда это положение вызывает споры. Некоторые возражают, что вследствие широкого распространения высокоскоростных сетей с высокой пропускной способностью распределение данных и функций управления ими потеряло смысл и проще хранить данные в центральном узле на очень большом компьютере и обращаться к ним по высокоскоростным сетям. Такой подход называется архитектурой с *вертикальным* масштабированием. Аргумент притягательный, но в нем упущена важная особенность распределенных баз данных. Во-первых, в большинстве современных приложений данные по факту распределенные; спорить можно только о том, как и где их обрабатывать. Во-вторых, и это даже важнее, в этой

аргументации не различается полоса пропускания сети (пропускная способность каналов связи) и задержка (сколько времени требуется для передачи данных). Задержка внутренне присуща распределенным средам, существуют физические ограничения на скорость передачи данных в компьютерных сетях. Удаленный доступ к данным может привести к задержкам, неприемлемым для многих приложений.

Второй момент – это то, что присущий распределенным системам параллелизм можно использовать для организации внутризапросного и межзапросного параллелизма. Под *межзапросным параллелизмом* понимается параллельное выполнение нескольких запросов, сгенерированных конкурентными транзакциями, с целью повысить транзакционную пропускную способность. Определение внутризапросного параллелизма различно в распределенных и параллельных СУБД. В первом случае внутризапросный параллелизм достигается путем разбиения одного запроса на несколько подзапросов, каждый из которых выполняется в отдельном узле и обращается к разным частям распределенной базы данных. В параллельных же СУБД это достигается за счет *межоператорного* и *внутриоператорного* параллелизма. Для реализации межоператорного параллелизма различные операторы в дереве запроса параллельно выполняются на разных процессорах, тогда как в случае внутриоператорного параллелизма один и тот же оператор выполняется несколькими процессорами, каждый из которых работает со своим подмножеством данных. Заметим, что обе эти формы параллелизма встречаются также в распределенной обработке запросов.

Внутриоператорный параллелизм основан на разложении одного оператора на множество независимых подоператоров, называемых *экземплярами оператора*. Это разложение производится с использованием секционирования отношений. Затем каждый экземпляр оператора обрабатывает одну секцию отношения. При разложении оператора часто удается воспользоваться начальным секционированием данных (например, если данные секционированы по атрибуту соединения). Для иллюстрации внутриоператорного параллелизма рассмотрим простой запрос SELECT с соединением. Оператор SELECT можно непосредственно разложить на несколько операторов SELECT, по одному для каждой секции, так что никакого перераспределения не требуется (рис. 1.6). Заметим, что если отношение секционировано по атрибуту выборки, то свойства секционирования можно использовать для исключения некоторых экземпляров SELECT. Например, в случае выборки по точному совпадению будет выполняться только один экземпляр SELECT, если отношение было секционировано по хешу (или по диапазону) атрибута выборки. Разложить оператор соединения JOIN сложнее. Чтобы соединения были независимы, каждую секцию одного отношения `R` можно соединить с другим отношением `S` целиком. Такое соединение будет крайне неэффективным (если только `S` не очень мало), потому что придется пересылать `S` на каждый участвующий в вычислении процессор. Более эффективно использовать свойства секционирования. Например, если `R` и `S` секционированы по хешу атрибута соединения и если имеет место операция эквисоединения, то мы можем секционировать соединение на независимые соединения. Этот идеальный случай, но он не всегда применим, потому что зависит от началь-

ного секционирования R и S. В других случаях один или два операнда, возможно, придется пересекционировать. Наконец, можно заметить, что функция секционирования (хеширование, по диапазону, циклически – все они обсуждаются в разделе 2.3.1) не зависит от локального алгоритма (например, вложенные циклы, хеширование, сортировка слиянием), используемого для обработки оператора соединения на каждом процессоре. Например, для соединения хешированием с использованием секционирования хешированием нужно две хеш-функции. Первая, h_1, применяется для секционирования двух базовых отношений по атрибуту соединения. Вторая же, h_2, может быть разной на каждом процессоре и применяется для обработки соединения на этом процессоре.

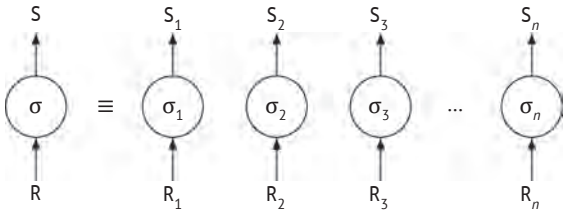

Рис. 1.6 ❖ Внутриоператорный параллелизм,
σ_i – *i*-й экземпляр оператора, *n* – степень параллелизма

Встречаются две формы межоператорного параллелизма. В случае *конвейерного параллелизма* несколько операторов, между которыми имеется связь вида производитель–потребитель, выполняются параллельно. Например, оба оператора выборки на рис. 1.7 будут выполняться параллельно с оператором соединения. Преимущество такого выполнения в том, что промежуточный результат необязательно материализовать целиком, поэтому экономится память и уменьшается число обращений к диску. *Независимый параллелизм* имеет место, когда между исполняемыми параллельно операторами нет никакой зависимости. Например, оба оператора выборки на рис. 1.7 можно выполнять параллельно. Такая форма параллелизма очень привлекательна, потому что между процессорами не возникает интерференции.

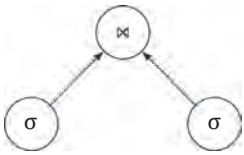

Рис. 1.7 ❖ Межоператорный параллелизм

1.4.4. Масштабируемость

В распределенной среде гораздо проще приспособиться к увеличению размера базы данных и рабочей нагрузки. Обычно систему расширяют путем

добавления в сеть новых процессоров и устройств хранения. Очевидно, что добиться линейного роста «мощности» не всегда возможно, потому что имеются накладные расходы на распределение. Но значительное улучшение все же достижимо. Именно поэтому распределенные СУБД занимают так много места в архитектурах с *горизонтальным* масштабированием в контексте кластерных и облачных вычислений. Под горизонтальным масштабированием понимают добавление новых серверов, не обремененных жесткими связями, что позволяет добиться почти бесконечной масштабируемости. Благодаря простоте добавления новых серверов баз данных распределенная СУБД может поддержать горизонтальное масштабирование.

1.5. Вопросы проектирования

В предыдущем разделе мы обсудили обещания технологии распределенных СУБД, отметив трудности, которые необходимо преодолеть для их реализации. В этом разделе мы продолжим обсуждение, описав проблемы, возникающие при проектировании распределенных СУБД. Этими вопросами мы будем заниматься до конца книги.

1.5.1. Проектирование распределенной базы данных

Вопрос заключается в том, как разместить данные в узлах. Отправной точкой является одна глобальная база данных, а конечным результатом – распределение данных между узлами. Такой подход называется *проектированием сверху вниз*. Есть два основных варианта размещения данных: *секционированное* (или *нереплицированное*) и *реплицированное*. В случае секционированной схемы база данных разбивается на несколько непересекающихся секций, каждая из которых размещается в отдельном узле. Репликация же может быть *полной* (говорят также о *полном дублировании*), когда в каждом узле размещается вся база, или *частичной* (*частичное дублирование*), когда каждая секция хранится в нескольких, но не во всех узлах. Два фундаментальных вопроса проектирования – *фрагментация*, т. е. разбиение базы данных на секции, называемые *фрагментами*, и *распределение*, т. е. оптимальное размещение фрагментов.

С этим связана проблема проектирования системного каталога и управления им. В централизованной СУБД *каталог* содержит метаинформацию о данных (т. е. их описание). В распределенной системе каталог содержит дополнительную информацию о местонахождении данных. Проблемы, связанные с управлением каталогом, по природе своей похожи на проблему размещения базы данных, рассмотренную в предыдущем разделе. Каталог может быть глобальным для всей распределенной СУБД или локальным в каждом узле; он может централизованно храниться в одном узле или распределяться между несколькими узлами; копий каталога может быть одна

или несколько. Проектирование и управление каталогом распределенной базы данных – тема главы 2.

1.5.2. Контроль распределенных данных

Важное требование к СУБД – обеспечение согласованности данных посредством управления доступа к ним. Это называется *контролем данными*, сюда относятся управление представлениями, контроль доступа и гарантии целостности. Распределенность влечет за собой дополнительные проблемы, потому что данные, для которых необходимо проверять правила, находятся в разных узлах, так что требуется распределенная проверка и предоставление гарантий. Эта тема рассматривается в главе 3.

1.5.3. Распределенная обработка запросов

Обработка запросов подразумевает проектирование алгоритмов, которые анализируют запрос и преобразуют его в последовательность операций манипулирования данными. Проблема заключается в нахождении стратегии наиболее эффективного выполнения запроса в сети при заданном определении стоимости. При этом нужно учитывать следующие факторы: распределение данных, затраты на связь, отсутствие достаточной информации, доступной локально. Цель – оптимизировать стоимость при вышеназванных ограничениях, используя внутренне присущий параллелизм для повышения производительности. По своей природе это NP-трудная задача, а подходы к ее решению обычно эвристические. Распределенная обработка запросов подробно обсуждается в главе 4.

1.5.4. Распределенное управление конкурентностью

Под управлением конкурентностью понимается синхронизация доступа к распределенной базе данных, обеспечивающая поддержание целостности данных. Постановки задачи управления конкурентностью в распределенном и централизованном контексте несколько различаются. Думать нужно не только о целостности одной базы данных, но и о согласованности нескольких ее копий. Требование, согласно которому значения нескольких копий каждого элемента данных должны в конечном итоге сходиться к одному и тому же значению, называется *взаимной согласованностью*.

Есть два широких класса решений: *пессимистические*, когда синхронизация выполнения запросов пользователей производится до начала выполнения, и *оптимистические*, когда запросы выполняются, а затем проверяется, не привело ли выполнение к нарушению согласованности. В обоих подходах используются два фундаментальных примитива: *блокировка*, основанная на взаимном исключении операций доступа к элементам данных, и *назначение*

временных меток, позволяющее упорядочить выполнение транзакций на основе присвоенных им временных меток. Существуют различные варианты этих схем, а также гибридные алгоритмы, пытающиеся сочетать оба базовых механизма.

В решениях на основе блокировки возможна взаимоблокировка, поскольку в разных транзакциях могут встречаться взаимно исключающие операции доступа к данным. Хорошо известные методы предотвращения, избегания и обнаружения-восстановления применимы и к распределенным СУБД. Распределенное управление конкурентностью обсуждается в главе 5.

1.5.5. Надежность распределенной СУБД

Выше мы говорили, что одно из потенциальных преимуществ распределенной системы – повышенная надежность и доступность. Но это не происходит автоматически. Важно предоставить механизмы, которые гарантируют согласованность базы данных, а также позволяют обнаруживать ошибки и восстанавливаться после них. Для распределенных СУБД это означает, что при возникновении отказа, когда некоторые узлы становятся неработоспособными или недоступными, базы данных в работающих узлах по-прежнему согласованы и актуальны. А когда вычислительная система или сеть восстанавливается после отказа, распределенная СУБД должна привести базы данных в неработавших узлах в актуальное состояние. Это особенно трудно в случае разделения сети, когда узлы распадаются на две или более групп, между которыми нет связи. Протоколы распределенной надежности – тема главы 5.

1.5.6. Репликация

Если распределенная база данных реплицирована (полностью или частично), то необходимо реализовать протоколы, обеспечивающие согласованность реплик, т. е. все копии одного и того же элемента данных должны иметь одинаковые значения. Эти протоколы могут быть *энергичными* (eager), т. е. обновления принудительно применяются ко всем репликам до завершения транзакции, или *ленивыми* (lazy), т. е. транзакция обновляет только одну копию (называемую *главной*), а потом изменения распространяются на другие копии уже после завершения транзакции. Протоколы репликации обсуждаются в главе 6.

1.5.7. Параллельные СУБД

Как уже было сказано, существует тесная связь между распределенными и параллельными базами данных. Хотя в первом случае предполагается, что каждый узел является отдельным логическим компьютером, на самом деле в большинстве установок узлы представляют собой параллельные кластеры. В этом заключается отмеченное ранее различие между распределением

в пределах одного узла, как в кластерных ЦОДах, и георазпределением. Цели, стоящие перед распределенными СУБД, отличаются от целей параллельных СУБД, основными из которых являются высокая масштабируемость и производительность. В этой книге основное внимание уделено управлению данными в георазпределенных базах данных, но при рассмотрении распределения в пределах одного узла, как в параллельной системе, возникает ряд интересных вопросов управления данными, которые мы обсудим в главе 8.

1.5.8. Интеграция баз данных

Одно из важных направлений развития – движение в сторону «более свободной» федерации источников данных, в т. ч. гетерогенных. В следующем разделе мы увидим, что это привело к развитию систем мультибаз данных (или *федеративных систем баз данных*), которые потребовали пересмотра некоторых фундаментальных методов работы с базами данных. Имеется множество уже распределенных баз данных, а цель – предоставить к ним простой доступ посредством их интеграции (физической или логической). Это требует *проектирования снизу вверх*. Такие системы составляют важную часть современных распределенных сред. В главе 7 мы обсудим системы мультибаз данных, или, как чаще говорят, *интеграцию баз данных*, в т. ч. вопросы проектирования и проблемы, возникающие при обработке запросов.

1.5.9. Альтернативные подходы к распределению

Рост интернета как фундаментальной сетевой платформы поднял важные вопросы, касающиеся предположений, лежащих в основе распределенных систем баз данных. Нам особенно интересны два из них: реинкарнация одноранговых вычислений, с одной стороны, и рост и развитие веба с другой. В обоих случаях конечная цель – усовершенствование разделения данных, но подходы разнятся, как и возникающие проблемы управления данными. В главе 9 мы обсудим одноранговое управление данными, а в главе 12 – управление веб-данными.

1.5.10. Обработка больших данных и NoSQL

В последние десять лет мы стали свидетелями взрывного роста обработки «больших данных». Точное определение больших данных дать трудно, но, как правило, все согласны с четырьмя характеристиками («четыре V»): данные очень большого объема (*volume*) разнородные (*variety*), обычно поступают с очень высокой скоростью (*velocity*) и могут быть низкокачественными в силу ненадежности источников и внутренних конфликтов (*veracity*). Предприняты значительные усилия для разработки систем работы с «большими данными», поскольку считается, что реляционные СУБД для многих приложений не подходят. Обычно эти усилия принимают одну из двух форм: приверженцы одной линии разрабатывают универсальные вычислительные

платформы (почти всегда горизонтально масштабируемые), а поклонники другой – специальные СУБД, не обладающие реляционными свойствами в полном объеме, но предлагающие более гибкие средства управления данными (так называемые системы NoSQL). Мы обсудим платформы больших данных в главе 10, а системы NoSQL – в главе 11.

1.6. Архитектуры распределенных СУБД

Архитектура системы определяет ее структуру. Это означает, что определены компоненты системы, функции каждой компоненты, взаимосвязи и взаимодействия между компонентами. Для спецификации архитектуры системы необходимо идентифицировать различные модули, их интерфейсы и взаимосвязи в терминах потока данных и управления в системе.

В этом разделе мы опишем четыре «эталонные» архитектуры[1] распределенной СУБД: клиент-серверная, одноранговая, мультибазовая и облачная. Это «идеализированные» взгляды на СУБД в том смысле, что многие коммерческие системы могут от них отклоняться; однако архитектуры служат разумной платформой, на которой можно обсуждать вопросы, относящиеся к распределенным СУБД.

Начнем с обсуждения пространства проектирования, чтобы лучше определить место архитектур, которые будут представлены ниже.

1.6.1. Архитектурные модели для распределенных СУБД

Мы пользуемся классификацией (рис. 1.8) с тремя осями, по которым можно охарактеризовать распределенные СУБД: (1) автономность локальных систем, (2) их распределение и (3) их гетерогенность. Все эти оси независимы друг от друга, и вдоль каждой из них имеется несколько вариантов. Следовательно, пространство проектирования состоит из 18 возможных архитектур. Не все они осмыслены, и большая их часть не имеет отношения к теме этой книги. На рис. 1.8 показаны три архитектуры, на которых мы сосредоточим внимание.

1.6.1.1. Автономность

В этом контексте слово *автономность* относится к распределению управления, а не данных. Речь идет о том, насколько независимо могут работать отдельные СУБД. Автономность зависит от многих факторов, в т. ч. от того, обмениваются ли системы-компоненты (т. е. отдельные СУБД) информацией, могут ли они независимо выполнять транзакции и разрешено ли вносить в них изменения.

---

[1] Эталонная архитектура обычно создается разработчиками стандартов с целью четко определить интерфейсы, подлежащие стандартизации.

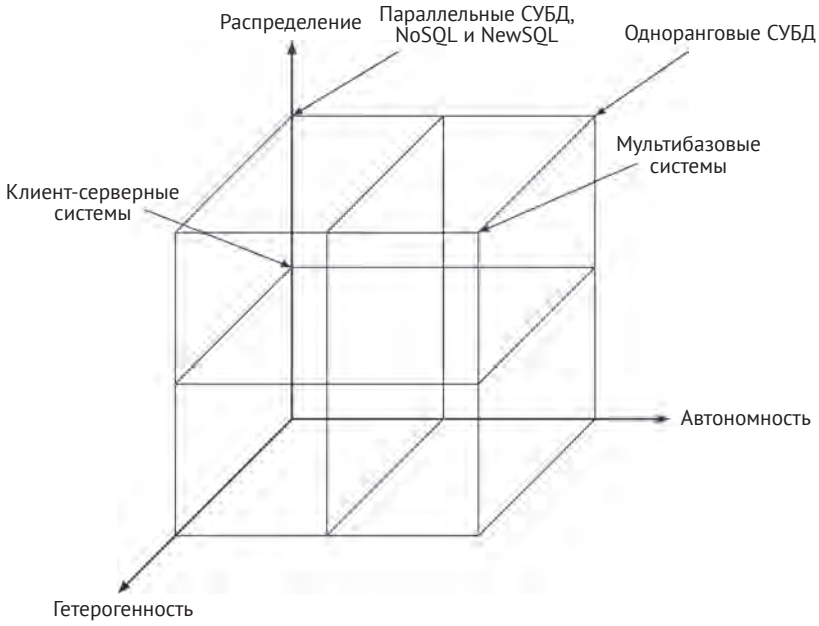

Рис. 1.8 ❖ Альтернативные реализации СУБД

Мы будем пользоваться классификацией, охватывающей важные аспекты указанных характеристик. В этой классификации выделяются три варианта. Первый – *тесная интеграция*, когда каждому пользователю, желающему обращаться к данным, которые могут находиться в нескольких базах, доступен единственный образ всей базы данных. С точки зрения пользователя, данные логически интегрированы в одной базе. В таких тесно интегрированных системах реализованы диспетчеры данных таким образом, что за обработку каждого запроса пользователя отвечает один диспетчер, даже если запрос обслуживается несколькими. Диспетчеры данных обычно не работают как независимые СУБД, пусть даже в них заложена соответствующая функциональность.

Далее мы выделяем *полуавтономные* системы, состоящие из СУБД, которые могут работать независимо (и обычно так и делают), но приняли решение участвовать в федерации, чтобы предоставить свои локальные данные в общее пользование. Каждая из этих СУБД решает, какие части управляемой ей базы сделать доступными пользователям других СУБД. Они не вполне автономны, т. к. для обмена информацией между собой нуждаются в модификации.

Последний вариант, который мы рассмотрим, – полная изоляция, когда отдельные системы – это самостоятельные СУБД, ничего не знающие ни о существовании других СУБД, ни о том, как с ними взаимодействовать. В таких системах обработка пользовательских транзакций, обращающихся к нескольким базам данных, особенно трудна, потому что нет никаких глобальных средств управления выполнением работы в отдельных СУБД.

1.6.1.2. Распределение

Если автономность относится к распределению (или децентрализации) управления, то ось распределения в нашей классификации имеет отношение к данным. Конечно, мы рассматриваем физическое распределение данных между несколькими узлами; как уже было сказано, пользователь видит данные как один логический пул. Существует много способов распределения СУБД. Мы выделили два больших класса: клиент-серверное и одноранговое (или *полное*) распределение. Вместе с отсутствием распределения получается три архитектуры по этой оси.

В случае клиент-серверного распределения управление данными сосредоточивается на серверах, а клиенты заняты обеспечением среды для приложения, в т. ч. для пользовательского интерфейса. Обязанности поддерживать коммуникацию возлагаются как на клиентские, так и на серверные компьютеры. Клиент-серверные СУБД – это практический компромисс в части функциональности распределения. Существует много способов их структуризации, и каждый из них предлагает разные уровни распределения. Детальное обсуждение мы отложим до раздела 1.6.2.

В одноранговых системах нет различия между клиентскими и серверными машинами. Каждая машина обладает всей функциональностью СУБД и может взаимодействовать с другими машинами для выполнения запросов и транзакций. В самом начале работы над распределенными системами баз данных основное внимание уделялось именно одноранговой архитектуре. Поэтому и в этой книге нас будут интересовать в первую очередь одноранговые (или *полностью распределенные*) системы, хотя многие из рассматриваемых методов переносятся также на клиент-серверные системы.

1.6.1.3. Гетерогенность

Гетерогенность может встречаться в распределенных системах в разных формах – от аппаратной гетерогенности и различий в сетевых протоколах до вариаций диспетчеров данных. С точки зрения этой книги, наиболее важны модели данных, языки запросов и протоколы управления транзакциями. Представление данных с помощью разных средств моделирования уже создает гетерогенность из-за различий в выразительной способности и ограничений разных моделей. Гетерогенность в области языков запросов не только включает использование совершенно разных парадигм доступа к данным в разных моделях (сравните доступ на уровне множеств в реляционных системах и на уровне записей в некоторых объектно-ориентированных системах), но и языковые различия даже в тех случаях, когда в разных системах используется одна и та же модель. Хотя в наши дни SQL стал стандартным реляционным языком запросов, существует много разных реализаций, и в диалекте от каждого производителя есть свои особенности. А на платформах больших данных и систем NoSQL вообще множество существенно различающихся языков и механизмов доступа.

1.6.2. Клиент-серверные системы

Клиент-серверные системы появились на сцене в начале 1990-х годов и оказали огромное влияние на технологию СУБД. Общая идея очень проста и элегантна: отличать функциональность, которую должна предоставлять серверная машина, от функциональности, ожидаемой от клиента. В результате создается *двухуровневая архитектура*, которая позволяет справляться со сложностью современных СУБД и распределения.

В реляционных клиент-серверных СУБД на сервер возлагается большая часть обязанностей по управлению данными. Вся обработка и оптимизация запросов, управление транзакциями и хранением производятся на сервере. Клиент, помимо прикладного и пользовательского интерфейса, включает *клиентский модуль СУБД*, отвечающий за управление данными, кешированными на стороне клиента, и иногда за управление транзакционными блокировками, которые тоже могут кешироваться. На стороне клиента можно также разместить проверку согласованности пользовательских запросов, но это делается нечасто, потому что требует репликации системного каталога на клиентских машинах. Эта архитектура, изображенная на рис. 1.9, типична для реляционных систем, в которых взаимодействие между клиентами и серверами производится на уровне SQL-команд. Иными словами, клиент передает SQL-запросы серверу, не пытаясь ни понять, ни оптимизировать их. Сервер делает большую часть работы и возвращает клиенту результирующее отношение.

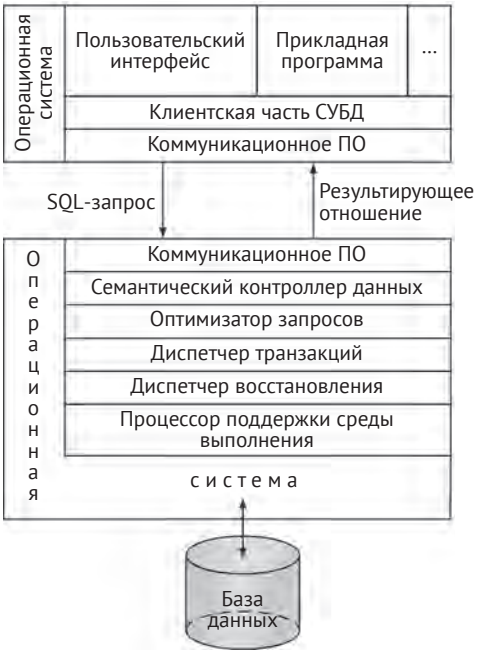

Рис. 1.9 ❖ Эталонная клиент-серверная архитектура

Имеется несколько разных реализаций клиент-серверной архитектуры. Простейший из них – когда существует единственный сервер, к которому обращается несколько клиентов. Этот вариант называется *несколько клиентов – один сервер*. С точки зрения управления данными, это не сильно отличается от централизованных баз данных, поскольку база хранится только на одной машине (сервере) вместе с программным обеспечением для управления ей. Однако существуют важные отличия от централизованных систем в плане выполнения транзакций и управления кешами – поскольку данные кешируются на стороне клиента, необходимо устанавливать протоколы когерентности кешей.

В более сложной клиент-серверной архитектуре имеется несколько серверов (подход *несколько клиентов – несколько серверов*). В этом случае возможны две альтернативные стратегии: либо каждый клиент сам управляет своим соединением с нужным ему сервером, либо клиент знает только о своем «домашнем сервере», который взаимодействует с другими серверами по мере необходимости. При первом подходе упрощается код сервера, но на клиентские машины возлагаются дополнительные обязанности. В результате получаются системы с «тяжелым клиентом». При втором подходе вся функциональность управления данными сосредоточивается на сервере. Таким образом, прозрачность доступа к данным обеспечивается на уровне интерфейса с сервером, а клиенты получаются «легкими».

В системах с несколькими серверами данные секционируются и могут реплицироваться между серверами. Если клиенты легкие, то весь этот механизм прозрачен для них, а серверы могут взаимодействовать друг с другом для ответа на запрос пользователя. Такой подход реализован в параллельных СУБД для повышения производительности посредством параллельной обработки.

Клиент-серверную архитектуру можно расширить, обеспечив более эффективное распределение функций между серверами разного типа: *клиенты* отрабатывают пользовательский интерфейс (например, веб-серверы), *серверы приложений* выполняют прикладные программы, а *серверы баз данных* отвечают за функции управления базой данных. Это приводит к трёхъярусной архитектуре распределенной системы.

Подход на основе серверов приложений (на самом деле *n*-ярусный подход к построению распределенных систем) можно обобщить, введя несколько серверов баз данных и несколько серверов приложений (рис. 1.10), и то же самое можно сделать в классических клиент-серверных архитектурах. В таком случае каждому серверу приложений обычно назначается одно или несколько приложений, а серверы баз данных работают в многосерверном режиме, рассмотренном выше. Кроме того, между приложением и сервером обычно устанавливается балансировщик нагрузки, который маршрутизирует запросы к подходящим серверам.

Подход на основе серверов баз данных, будучи обобщением классической клиент-серверной архитектуры, имеет несколько потенциальных преимуществ. Во-первых, концентрация управления данными в одном месте позволяет разрабатывать специальные методы для повышения надежности и доступности, например с помощью параллелизма. Во-вторых, общую про-

изводительность управления базами данных можно значительно улучшить благодаря тесной интеграции системы баз данных со специализированной для управления базами данных операционной системой. Наконец, серверы баз данных могут пользоваться дополнительным оборудованием, например графическими процессорами и ППВМ, чтобы повысить производительность и доступность данных.

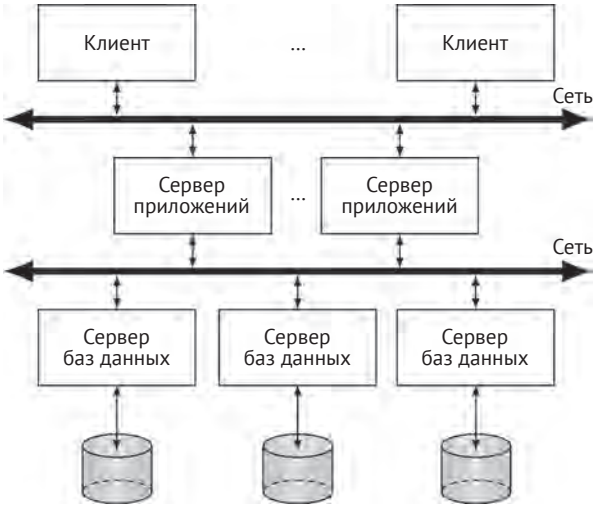

Рис. 1.10 ❖ Серверы распределенной базы данных

Хотя эти преимущества значительны, имеются дополнительные накладные расходы на еще один уровень взаимодействия между серверами приложений и серверами баз данных. Стоимость взаимодействия можно амортизировать, если интерфейс сервера достаточно высокоуровневый и допускает выражение сложных запросов, включающих интенсивную обработку данных.

1.6.3. Одноранговые системы

Все ранние работы по распределенным СУБД были сконцентрированы на одноранговых архитектурах, в которых функциональность всех узлов системы одинакова. У современных одноранговых систем есть два важных отличия от предшественников. Во-первых, это массовое распределение. Если раньше речь шла о нескольких (от силы десяти) узлах, то нынешние системы насчитывают тысячи узлов. Во-вторых, это внутренне присущая всем аспектам узлов гетерогенность и автономность узлов. Хотя, как мы уже обсуждали, это всегда представляло интерес для распределенных баз данных, в сочетании с массовым распределением гетерогенность и автономность узлов приобрели дополнительную важность, исключив из рассмотрения некоторые подходы. В этой книге мы сначала сосредоточимся на классическом

понимании одноранговости (одинаковая функциональность всех узлов), поскольку принципы и фундаментальные методы таких систем очень близки к клиент-серверным системам, а в главе 9 отдельно обсудим современные одноранговые системы.

В этих системах проектирование базы данных ведется сверху вниз, как было описано выше. То есть на входе имеется централизованная база данных со своим определением схемы (*глобальная концептуальная схема* – ГКС). Эта база данных секционируется и размещается в узлах распределенной СУБД. Таким образом, в каждом узле появляется локальная база данных с собственной схемой (называемой *локальной концептуальной схемой* – ЛКС). Пользователь формулирует запросы, ориентируясь на ГКС, независимо от ее местоположения. Распределенная СУБД транслирует глобальные запросы во множество локальных запросов, которые выполняются взаимодействующими между собой компонентами распределенной СУБД в разных узлах. С точки зрения обработки запросов, одноранговые системы и клиент-серверные СУБД предлагают одинаковое представление данных. То есть у пользователя создается впечатление логически единой базы данных, хотя на физическом уровне данные распределены.

Компоненты распределенной СУБД показаны на рис. 1.11. Один компонент отвечает за взаимодействие с пользователями, а другой – за хранение. Первый главный компонент, который мы назвали *процессором пользователей*, состоит из четырех элементов.

Первая главная компонента, которую мы назвали *процессором пользователей*, состоит из четырех элементов.

1. *Обработчик пользовательского интерфейса* отвечает за интерпретацию команд пользователя и форматирование результатов, отправляемых пользователю.
2. Контроллер данных использует определенные в глобальной концептуальной схеме ограничения целостности и разрешения, для того чтобы проверить, можно ли выполнить запрос пользователя. Эта компонента, которую мы будем подробно изучать в главе 3, также отвечает за авторизацию и другие функции.
3. *Глобальный оптимизатор запросов* определяет стратегию выполнения, минимизирующую функцию стоимости, и преобразует глобальные запросы в локальные, пользуясь для этого глобальной и локальной концептуальными схемами, а также глобальным каталогом. Среди прочего, глобальный оптимизатор запросов отвечает за генерирование наилучшей стратегии выполнения распределенных операций соединения. Эти вопросы обсуждаются в главе 4.
4. *Монитор распределенного выполнения* координирует распределенное выполнение пользовательских запросов. Он также называется *диспетчером распределенных транзакций*. При распределенном выполнении запросов мониторы выполнения, находящиеся в разных узлах, могут взаимодействовать между собой, и обычно так и происходит. Функциональность монитора распределенных транзакций описывается в главе 5.

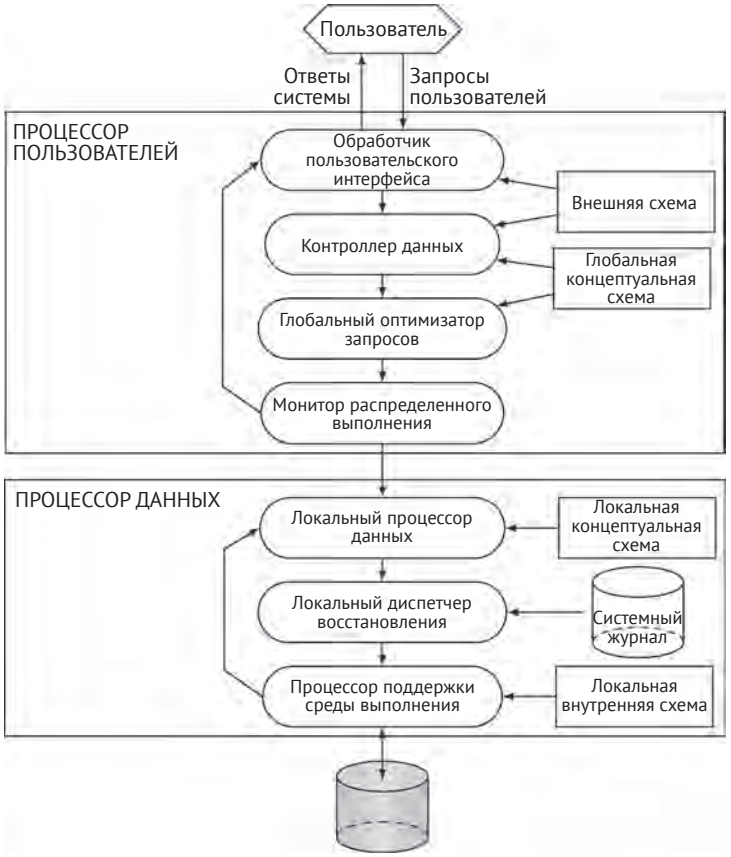

Рис. 1.11 ❖ Компоненты распределенной СУБД

Вторая главная компонента распределенной СУБД – *процессор данных* – состоит из следующих трех элементов. Все они решаются централизованной СУБД, так что в этой книге мы их рассматривать не будем.

1. *Локальный оптимизатор запросов*, который фактически играет роль селектора пути доступа и отвечает за выбор лучшего пути доступа к элементам данным[1].
2. Локальный диспетчер восстановления отвечает за согласованность локальной базы данных даже после сбоев.
3. Процессор поддержки среды выполнения осуществляет физический доступ к базе данных в соответствии с командами, включенными оптимизатором запросов в план выполнения. Этот процессор реализует интерфейс с операционной системой и содержит диспетчер буфера базы данных (кеша), который отвечает за поддержание буфера в основной памяти и управление доступом к данным.

[1] Термин *путь доступа* относится к структурам данных и алгоритмам, используемым для доступа к данным. Например, типичным путем доступа является индекс, построенный по одному или нескольким атрибутам отношения.

Важно отметить, что наше употребление терминов «процессор пользователей» и «процессор данных» не подразумевает функционального деления, как в клиент-серверных системах. Это деление по преимуществу организационное и не предполагает, что компоненты должны размещаться на разных машинах. В одноранговых системах модули процесса пользователей и процессора данных обычно находятся на каждой машине. Но бывают «узлы запросов», в которых присутствует только процессор пользователей.

1.6.4. Системы управления мультибазами данных

В системах управления мультибазами данных (СУМБД), или мультибазовых системах, отдельные СУБД полностью автономны и понятия кооперации не существует; они могут даже не знать о существовании друг друга и не иметь никаких средств взаимодействия. Естественно, нас интересуют прежде всего распределенные СУМБД, т. е. такие, в которых составляющие СУБД размещены в разных узлах. Многие из рассмотренных нами вопросов являются общими для одноузловых и распределенных СУМБД; в таких случаях мы будем просто говорить «СУМБД» без указания природы. В современной литературе чаще встречается термин *интеграция баз данных*. Мы будем обсуждать такие системы в главе 7. Заметим, однако, что термину «мультибаза» в разных источниках приписывается различный смысл. В этой книге мы будем понимать под ним сказанное выше, но следует помнить, что такая трактовка может не совпадать с встречающейся в литературе.

Различия в уровне автономности СУМБД и распределенных СУБД находят отражение и в их архитектурных моделях. Фундаментальное различие связано с определением глобальной концептуальной схемы. В случае логически интегрированных распределенных СУБД глобальная концептуальная схема определяет концептуальное представление *всей* базы данных, тогда как в случае СУМБД она представляет только набор *некоторых* локальных баз данных, которые локальная СУБД готова предоставить в общее пользование. Отдельные СУБД могут предоставить общий доступ только к части своих данных. Поэтому определения *глобальной базы данных* в СУМБД и распределенных СУБД различаются. Во втором случае глобальная база данных совпадает с объединением локальных, а в первом она является лишь подмножеством (быть может, собственным) этого объединения. В СУМБД глобальная концептуальная схема (иногда называемая *опосредованной схемой*) определяется путем интеграции локальных концептуальных схем (или их частей).

Компонентная архитектурная модель распределенной СУМБД существенно отличается от модели распределенной СУБД тем, что каждый узел полноценной СУБД управляет своей базой данных. СУМБД предоставляет слой программного обеспечения, работающий поверх этих отдельных СУБД и предлагающий пользователям средства доступа к разным базам данных (рис. 1.12). Отметим, что в распределенной СУМБД слой СУМБД может работать в нескольких узлах или находиться в одном узле, который и предоставляет соответствующие службы. Также заметим, что, с точки зрения отдель-

ных СУБД, слой СУМБД является просто еще одним приложением, которое отправляет запросы и получает ответы.

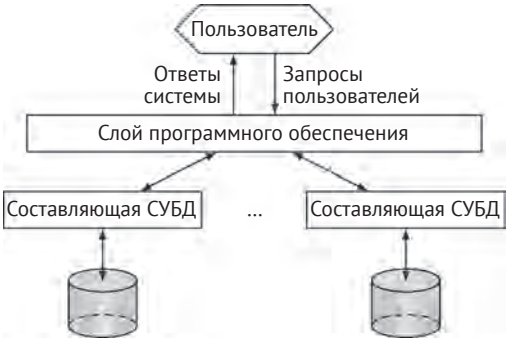

Рис. 1.12 ❖ Компоненты СУМБД

Популярной реализацией архитектуры СУМБД является подход на основе посредников и оберток (рис. 1.13). *Посредником* называется «программный модуль, который использует закодированные знания о некоторых множествах или подмножествах данных для порождения информации, предназначенной более высоким уровням приложений» [Wiederhold 1992]. Таким образом, каждый посредник выполняет конкретную функцию с четко определенным интерфейсом. При такой архитектуре каждый модуль, принадлежащий слою СУМБД на рис. 1.12, реализован в виде посредника. Поскольку посредники могут быть надстроены над другими посредниками, можно организовать многоуровневую реализацию. Уровень посредников реализует ГКС. Именно на этом уровне обрабатываются пользовательские запросы к ГКС и предоставляется функциональность СУМБД.

Посредники обычно работают с общей моделью данных и языком определения интерфейса. Чтобы справиться с потенциальной гетерогенностью СУБД-источников, создаются *обертки*, задача которых – предоставить отображение между СУБД-источником и посредником. Например, если СУБД-источник реляционный, а посредник объектно-ориентированный, то необходимые отображения реализуются обертками. Точная роль и функция посредников зависят от реализации. В некоторых случаях посредники не делают ничего, кроме трансляции, тогда они называются «тонкими». Но иногда посредники берут на себя часть функциональности обработки запросов.

Набор посредников можно рассматривать как промежуточный уровень, предоставляющий службы, надстроенные над системами-источниками. ПО промежуточного уровня – тема, которая активно разрабатывалась в прошлом десятилетии, когда были созданы весьма развитые системы, предлагающие продвинутые средства для разработки распределенных приложений. Обсуждаемые нами посредники – лишь часть функциональности таких систем.

Рис. 1.13 ❖ Архитектура с посредниками/обертками

1.6.5. Облачные вычисления

Облачные вычисления привели к тектоническим сдвигам в методах развертывания масштабируемых приложений пользователями и организациями, а в особенности приложений для управления данными. Видится система (обычно она изображается в виде облака), предоставляющая через интернет по запросу надежные службы с простым доступом к практически бесконечным ресурсам – вычислительным, хранения и сетевым. Пользуясь очень простыми веб-интерфейсами и за скромную плату, пользователи могут выносить такие сложные задачи, как хранение данных, управление базами данных, администрирование системы или развертывание приложений за пределы предприятия, в очень крупные центры обработки данных, эксплуатируемые поставщиками облачных служб. Таким образом, сложность управления программно-аппаратной инфраструктурой переходит от пользовательской организации к облачному поставщику.

Облачные вычисления – это естественная эволюция и сочетание различных моделей вычислений, предложенных для поддержки приложений через веб: сервисно-ориентированных архитектур (SOA) для высокоуровневого взаимодействия приложений с помощью веб-служб, служебных вычислений для упаковки вычислительных ресурсов и ресурсов хранения в виде служб, кластерных технологий и виртуализации для управления многочисленными ресурсами (вычислительными и хранения) и автономных вычислений для самоконтроля сложной инфраструктуры. Облако предоставляет различные уровни функциональности:

- инфраструктура как услуга (IaaS): предложение вычислительной инфраструктуры (вычисления, сеть и ресурсы хранения) в виде услуги;
- платформа как услуга (PaaS): предложение вычислительной платформы вместе с инструментами разработки и API в виде услуги;
- программное обеспечение как услуга (SaaS): предложение прикладного ПО в виде услуги;
- база данных как услуга (DaaS): предложение базы данных в виде услуги.

Уникальность облачных вычислений заключается в предоставлении таких комбинаций услуг, которые наилучшим образом отвечают требованиям пользователя. С технической точки зрения, проблема в том, чтобы сравнительно дешево поддержать очень большую инфраструктуру, способную управлять множеством пользователей и ресурсов, обеспечивая высокое качество обслуживания.

Дать точное определение облачных вычислений, с которым все были бы согласны, трудно, т. к. точек зрения много (коммерческая, маркетинговая, техническая, исследовательская и т. д.). Но можно принять рабочее определение – «облако по запросу предоставляет через интернет ресурсы и услуги, обычно сравнимые по масштабу и надежности с центром обработки данных» [Grossman and Gu 2009]. В этом определении отражена главная цель (предоставление ресурсов и услуг по запросу через интернет) и основные требования к технической поддержке (сравнимые по масштабу и надежности с центром обработки данных). Поскольку доступ к ресурсам осуществляется посредством служб, то пользователь получает все запрошенное в виде услуг. Поэтому, как и в сфере услуг, облачные поставщики могут работать по модели ценообразования «плата по факту», когда пользователь платит только за потребленные ресурсы.

Основными функциями, предоставляемыми облаками, являются: безопасность, управление каталогом, управление ресурсами (подготовка, выделение, мониторинг) и управление данными (хранение, управление файлами, управление базой данных, репликация данных). Кроме того, облако реализует начисление платы, бухгалтерский учет и гарантии соглашения об уровне обслуживания (SLA). Перечислим типичные преимущества облачных вычислений.

- **Стоимость.** Затраты заказчика заметно сокращаются, поскольку ему не нужно нести расходы на владение и управление инфраструктурой; счета выставляются только за потребленные ресурсы. С точки зрения облачного поставщика, поддержание консолидированной инфраструктуры и отнесение затрат на многих заказчиков уменьшает стоимость владения и эксплуатации.
- **Простота доступа и использования.** Облако скрывает сложность ИТ-инфраструктуры и обеспечивает прозрачность местоположения и распределения. Поэтому заказчик может получить доступ к ИТ-услугам в любое время и из любого места, где можно подключиться к интернету.
- **Качество обслуживания.** Эксплуатация ИТ-инфраструктуры специализированным поставщиком, который имеет обширный опыт работы с очень крупной инфраструктурой (в т. ч. своей собственной), повышает качество обслуживания и операционную эффективность.

- **Инновационность.** Использование самых свежих инструментов и приложений, предлагаемых облаком, способствует внедрению современных технологий, повышая тем самым способность заказчиков к восприятию инноваций.
- **Эластичность.** Способность к динамическому вертикальному масштабированию (вверх и вниз) для адаптации к изменяющимся условиям – важное преимущество. Обычно это достигается посредством виртуализации серверов – технологии, которая позволяет нескольким приложениям работать в виртуальных машинах (ВМ) на одном физическом компьютере, т. е. как если бы они работали на физически разных компьютерах. Тогда заказчик может затребовать вычислительные экземпляры в виде ВМ и присоединять к ним ресурсы хранения по мере необходимости.

Однако имеются и недостатки, которые нужно хорошо понимать, принимая решение о переходе в облако. Они такие же, как при передаче приложений и данных в аутсорсинг внешней компании.

- **Зависимость от поставщика.** Заказчик, как правило, оказывается прочно привязан к облачному поставщику, поскольку использует его закрытое программное обеспечение и форматы, а стоимость передачи данных за пределы системы поставщика высока. Все это сильно затрудняет миграцию из облачной службы.
- **Потеря контроля.** Заказчик может потерять административный контроль над критическими операциями, в т. ч. временем простоя системы, например для перехода на новую версию ПО.
- **Безопасность.** Поскольку доступ к облачным данным возможен из любого места, где есть интернет, атаки могут скомпрометировать данные компании. Безопасность облака можно повысить с помощью дополнительных средств, например путем организации виртуального частного облака, но иногда их трудно интегрировать с политикой безопасности компании.
- **Скрытые затраты.** Модификация приложения для работы в облаке типа SaaS или PaaS может потребовать весьма дорогостоящих доработок.

Не существует стандарта облачной архитектуры, и, по-видимому, он никогда не появится, потому что разные облачные поставщики предлагают разные облачные службы (IaaS, PaaS, SaaS и т. д.) разными способами (публичное, частное, виртуальное частное и т. д.), зависящими от бизнес-модели. Поэтому мы обсудим упрощенную облачную архитектуру с упором на управление базами данных.

Как правило, облако включает несколько территориально разнесенных узлов, или ЦОДов (рис. 1.14), каждый со своими ресурсами и данными. Крупные облачные поставщики разделяют весь мир на несколько регионов, в каждом из которых находится несколько узлов. Тому есть три основные причины. Во-первых, задержка в регионе пользователя низкая, потому что запросы поступают ближайшему узлу. Во-вторых, репликация данных между узлами в разных регионах обеспечивает высокую доступность и, в частности, устойчивость к катастрофическим отказам узлов. В-третьих, законы некоторых стран о защите персональных данных пользователей вынуждают облачных поставщиков располагать ЦОДы в соответствующем регионе (например, в Ев-

ропе). Многоузловая прозрачность обычно является режимом по умолчанию, т. е. облако представляется «централизованным», а поставщик услуг может оптимизировать выделение ресурсов пользователям. Однако некоторые облачные поставщики (в частности, Amazon и Microsoft) делают узлы видимыми пользователям (или разработчикам приложений). Это позволяет либо выбрать конкретный ЦОД для установки приложения вместе с его базой данных, либо развернуть очень большое приложение в нескольких узлах, взаимодействующих посредством веб-служб. Например, глядя на рис. 1.14, можно представить, что Клиент 1 сначала подключается к приложению в ЦОДе 1, которое затем обращается к приложению в ЦОДе 2 через веб-службу.

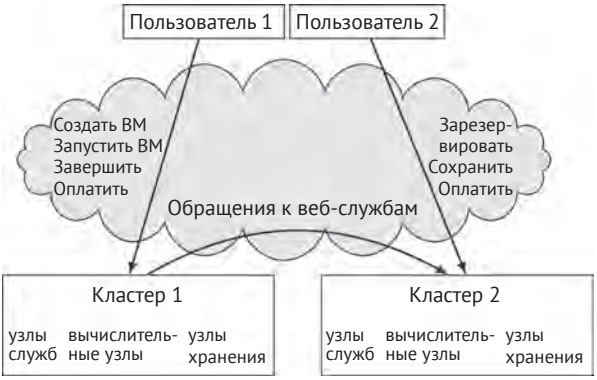

Рис. 1.14 ❖ Упрощенная архитектура облака

Архитектура облачного узла (центра обработки данных) обычно трехъярусная. На первом ярусе располагаются веб-клиенты, которые обращаются к облачным веб-серверам, как правило, пользуясь маршрутизатором или балансировщиком нагрузки в облачном узле. На втором ярусе располагаются веб-серверы и серверы приложений, которые работают с клиентами и предоставляют бизнес-логику. Третий уровень включает серверы баз данных. Возможны и серверы других типов, например для кеширования данных между серверами приложений и баз данных. Таким образом, облачная архитектура предлагает два уровня распределения: территориальное распределение между узлами на основе глобальной сети и распределение между серверами на одном узле, обычно образующими кластер компьютеров. На первом уровне применяются методы территориально распределенных СУБД, а на втором – методы параллельных СУБД.

Облачные вычисления изначально были спроектированы веб-гигантами для выполнения своих приложений очень крупного масштаба в ЦОДах, насчитывающих тысячи серверов. Системы обработки больших данных (глава 10) и системы NoSQL/NewSQL (глава 11) специально «заточены» под требования таких приложений в облаке и используют методы управления распределенными данными. С появлением решений типа SaaS и PaaS облачным поставщиком также понадобилось предоставлять очень большому числу заказчиков, называемых *арендаторами*, небольшие приложения, каждое из

которых ведет собственную (небольшую) базу данных, к которой обращаются его пользователи. Выделять отдельный сервер каждому арендатору слишком расточительно в плане аппаратных ресурсов. Чтобы уменьшить непроизводительное расходование ресурсов и снизить операционные затраты, облачные поставщики обычно организуют разделение ресурсов между арендаторами, применяя «многоарендные» архитектуры, в которых один сервер может поддерживать несколько арендаторов. В различных моделях многоарендности приняты разные компромиссы между производительностью, изоляцией (в плане безопасности и производительности) и сложностью проектирования. В IaaS применяется простая модель совместного использования оборудования, которая обычно реализуется с помощью виртуализации серверов, когда под каждую арендуемую базу данных и операционную систему отводится своя ВМ. Эта модель обеспечивает высокую изоляцию с точки зрения безопасности. Но степень совместного использования ресурсов ограничена из-за наличия избыточных экземпляров СУБД (по одному на ВМ), которые не взаимодействуют между собой и управляют ресурсами независимо. В контексте SaaS, PaaS или DaaS можно выделить три основные модели многоарендного управления базами данных, в которых степень разделения ресурсов и производительности увеличивается, но за счет меньшей изоляции и большей сложности.

- **Разделяемый сервер СУБД.** В этой модели арендаторы разделяют сервер с общим экземпляром СУБД, но база данных у каждого арендатора своя. Большинство СУБД предлагают поддержку нескольких баз данных одним экземпляром сервера. Поэтому такую модель легко поддержать. Она обеспечивает строгую изоляцию на уровне базы данных и более эффективна, чем совместное использование оборудования, поскольку экземпляр СУБД осуществляет полный контроль аппаратных ресурсов. Однако раздельное управление каждой базой данных все же ведет к неэффективному управлению ресурсами.
- **Разделяемая база данных.** В этой модели арендаторы разделяют одну базу данных, но у каждого имеется своя схема и набор таблиц. Консолидация баз данных обычно обеспечивается дополнительным уровнем абстракции внутри СУБД. Эта модель реализована в некоторых СУБД (например, Oracle) с помощью одной контейнерной базы, содержащей несколько баз данных. Она предлагает эффективное использование ресурсов и хорошую изоляцию на уровне схемы. Однако при наличии большого числа (тысяч) арендаторов сервера образуется много мелких таблиц, что ведет к заметным накладным расходам.
- **Разделяемые таблицы.** В этой модели арендаторы разделяют базу данных, схему и таблицы. Чтобы понять, какому арендатору принадлежит конкретная строка таблицы, обычно заводят дополнительный столбец tenant_id. Хотя при этом достигается лучшая степень разделения ресурсов (например, кеша в памяти), изоляция ниже – как в плане безопасности, так и в плане производительности. Например, крупным заказчикам будет принадлежать больше строк в разделяемых таблицах, из-за чего страдает производительность заказчиков поменьше.

1.7. Библиографические замечания

По распределенным СУБД не так много книг. Две из числа первых, Ceri and Pelagatti [1983] и Bell and Grimson [1992], больше не издаются. В более поздней книге Rahimi and Haug [2010] изложены некоторые классические вопросы, затрагиваемые и в этой книге. Кроме того, почти в каждой книге по базам данных есть глава, посвященная распределенным СУБД.

Первопроходческие системы Distributed INGRES и SDD-1 обсуждаются в работах [Stonebraker and Neuhold 1977] и [Wong 1977] соответственно.

Введение в проектирование баз данных имеется в работе [Levin and Morgan 1975], а более полно рассматривается в работе [Ceri et al. 1987]. Обзор алгоритмов распределенного хранения файлов см. в статье [Dowdy and Foster 1982]. Управление каталогами не получило подробного освещения в научном сообществе, но об общих методах можно прочитать в работах [Chu and Nahouraii 1975] и [Chu 1976]. Обзор методов обработки запросов см. в работе [Sacco and Yao 1982]. Описание алгоритмов управления конкурентностью приведено в работах [Bernstein and Goodman 1981] и [Bernstein et al. 1987]. Управление взаимоблокировками также является темой обширных исследований; введение имеется в статье [Isloor and Marsland 1980], но чаще цитируется работа [Obermack 1982]. Хорошими обзорами по методам обнаружения взаимоблокировок являются работы [Knapp 1987] и [Elmagarmid 1986]. Надежность относится к числу вопросов, обсуждаемых в работе [Gray 1979], одной из основополагающих в этой области. Из других важных работ на эту тему упомянем [Verhofstadt 1978] и [Härder and Reuter 1983]. Работа [Gray 1979] также является первой, в которой обсуждаются вопросы поддержки распределенных баз данных со стороны операционной системы; та же тема рассматривается в работе [Stonebraker 1981]. К сожалению, в обеих работах на первом плане стоят централизованные базы данных. Очень хороший ранний обзор систем управления мультибазами данных см. в работе Sheth and Larson [1990], а в статье Wiederhold [1992] предложен подход к СУМБД на основе посредников и оберток. Облачные вычисления – тема огромного количества недавно вышедших книг; в качестве хорошей отправной точки назовем книгу [Agrawal et al. 2012], а в работе [Cusumano 2010] приведен неплохой краткий обзор. Архитектура, рассмотренная в разделе 1.6.5, заимствована из работы [Agrawal et al. 2012]. Различные многоарендные модели в облачных средах обсуждаются в работах [Curino et al. 2011] и [Agrawal et al. 2012].

Существует немало предложений по выбору архитектуры. Из наиболее интересных назовем работу [Schreiber 1977], в которой подробно описано расширение системы ANSI/SPARC как попытка включить гетерогенность моделей данных, а также предложение Mohan and Yeh [1978]. Понятно, что они относятся к самому началу развития технологии распределенных СУБД. Детальная компонентная архитектура системы, приведенная на рис. 1.11, заимствована из работы [Rahimi 1987]. Альтернатива классификации, показанной на рис. 1.8, приведена в работе [Sheth and Larson 1990].

В книге [Agrawal et al. 2012] великолепно изложены проблемы и концепции управления данными в облаке, в т. ч. распределенные транзакции, системы обработки больших данных и многоарендные базы данных.